AFRL-AFOSR-VA-TR-2023-0317

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Salvucci, Dario
DREXEL UNIVERSITY
3141 CHESTNUT ST
PHILADELPHIA, PA, 19104
USA

03/22/2023
Final Technical Report

Air Force Research Laboratory
Air Force Office of Scientific Research
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
20230322	Final	START DATE 20180801	END DATE 20210731

4. TITLE AND SUBTITLE
Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
	FA9550-18-1-0371	61102F
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Dario Salvucci

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DREXEL UNIVERSITY
3141 CHESTNUT ST
PHILADELPHIA, PA 19104
USA

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Office of Scientific Research 875 N. Randolph St. Room 3112 Arlington, VA 22203	AFRL/AFOSR RTA2	AFRL-AFOSR-VA-TR-2023-0317

12. DISTRIBUTION/AVAILABILITY STATEMENT
A Distribution Unlimited: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity. The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a “cognitive code” cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	33

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
LAURA STECKMAN	426-7556

Standard Form 298 (Rev.5/2020)
Prescribed by ANSI Std. Z39.18

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

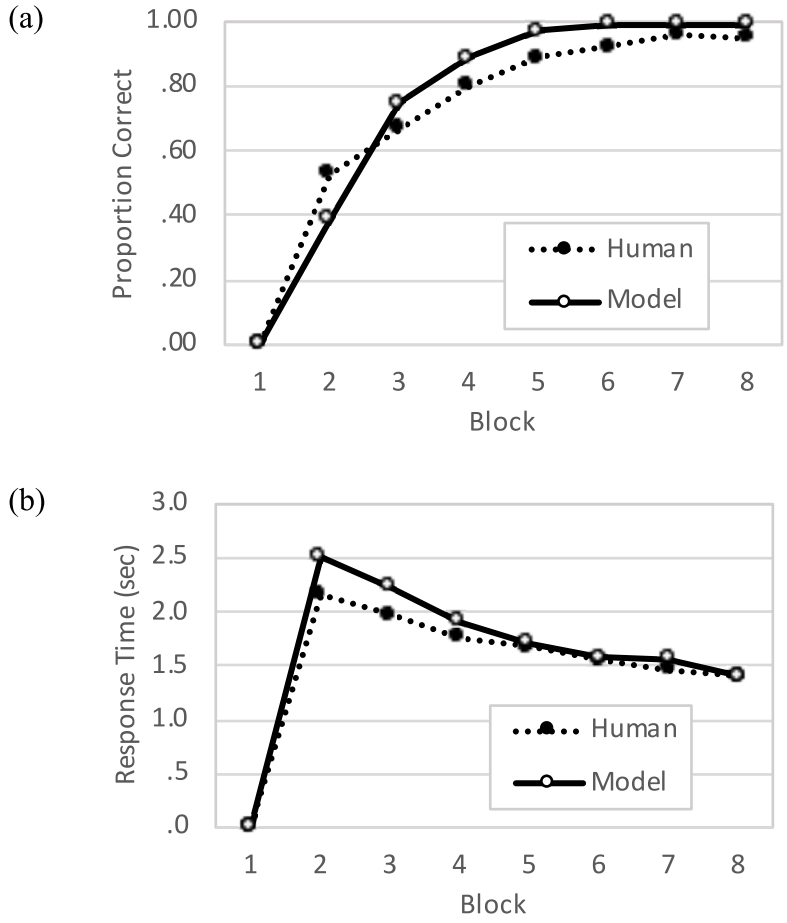
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

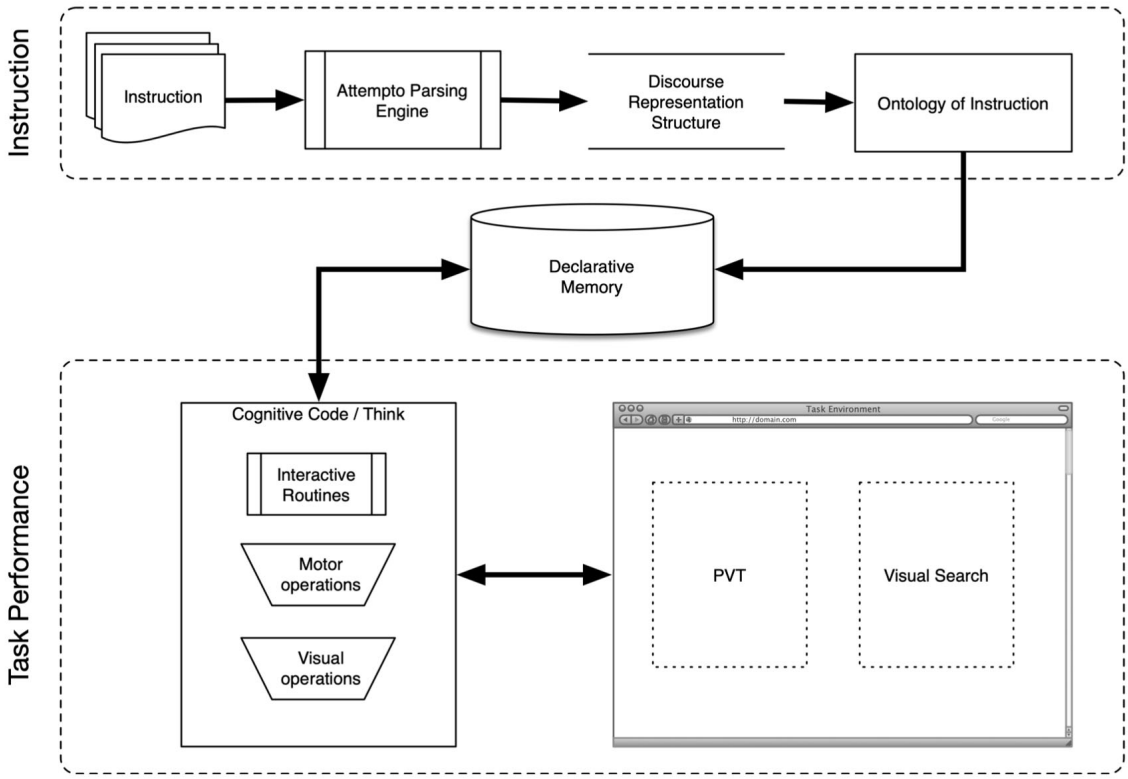


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

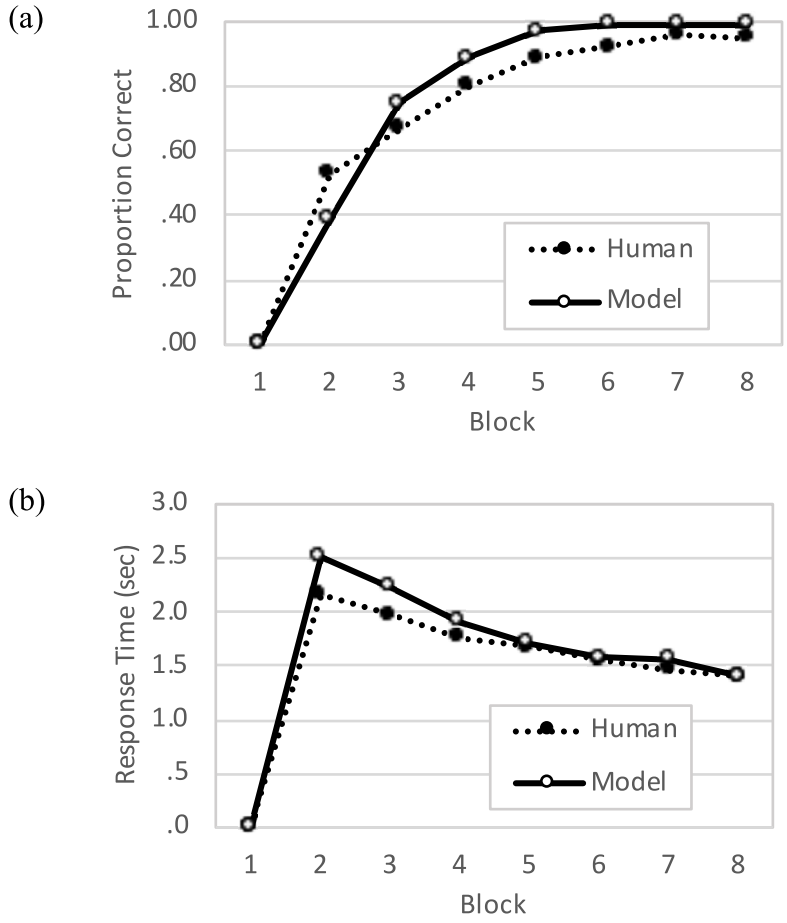
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

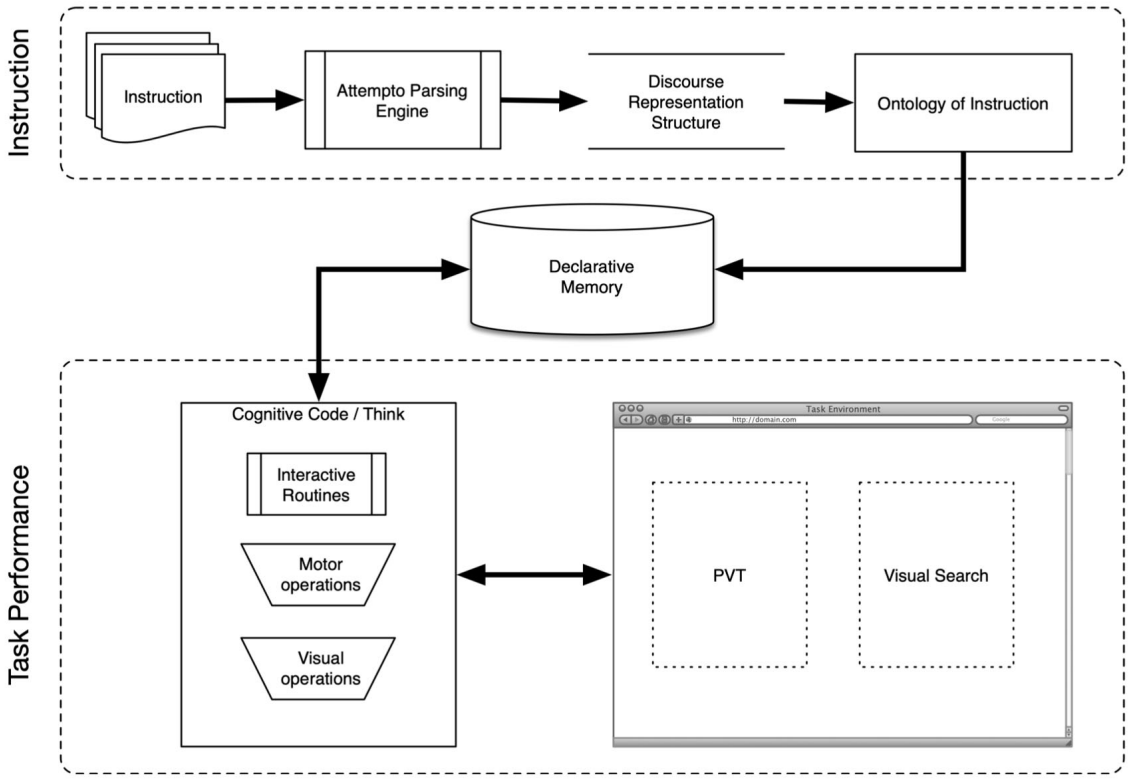


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

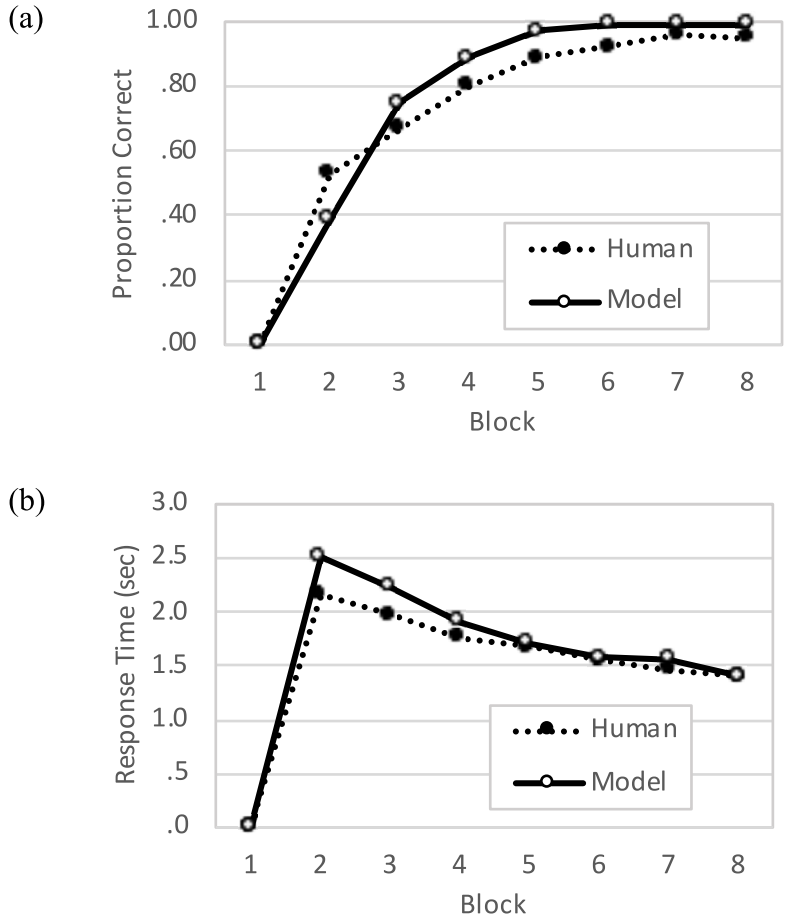
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

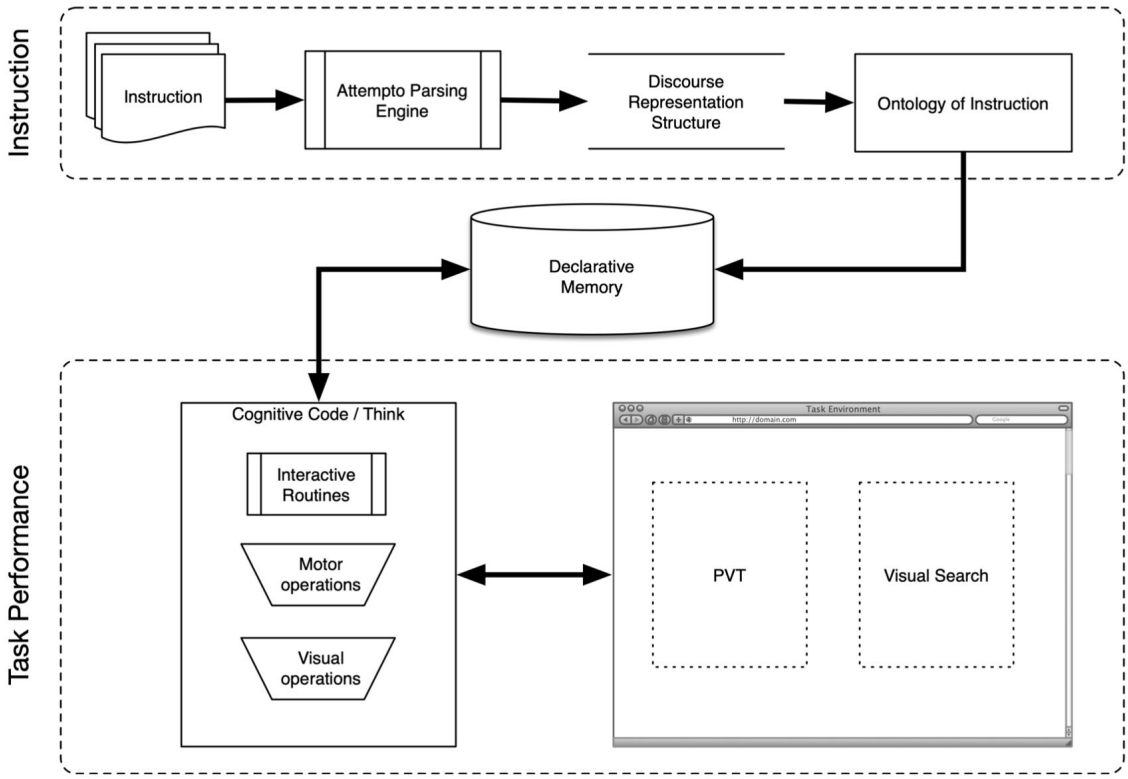


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

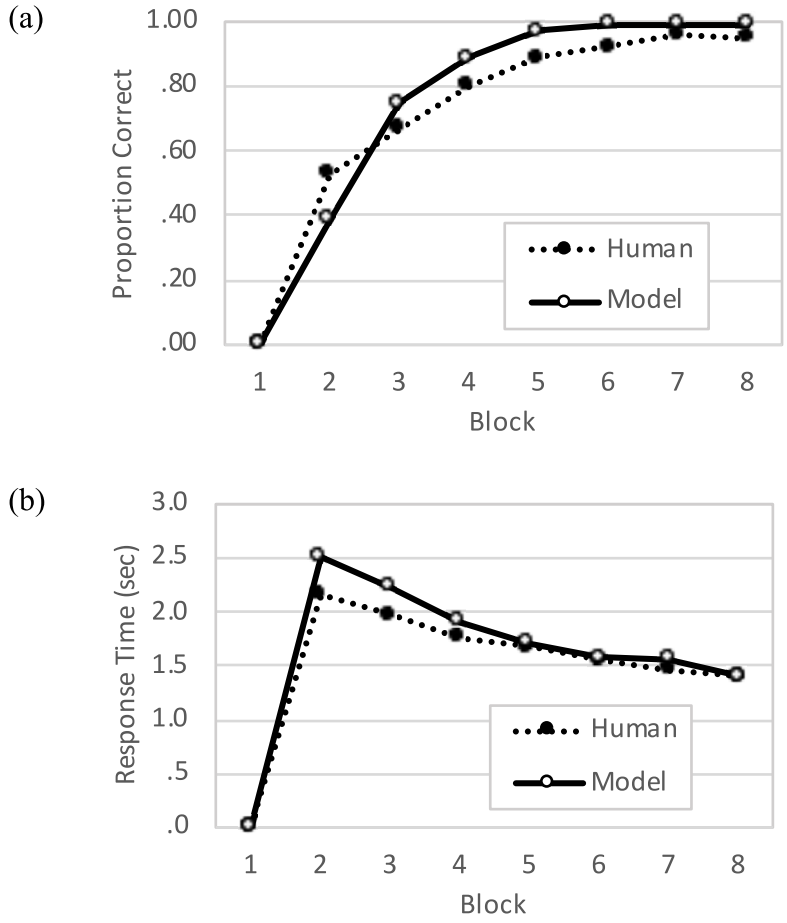
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

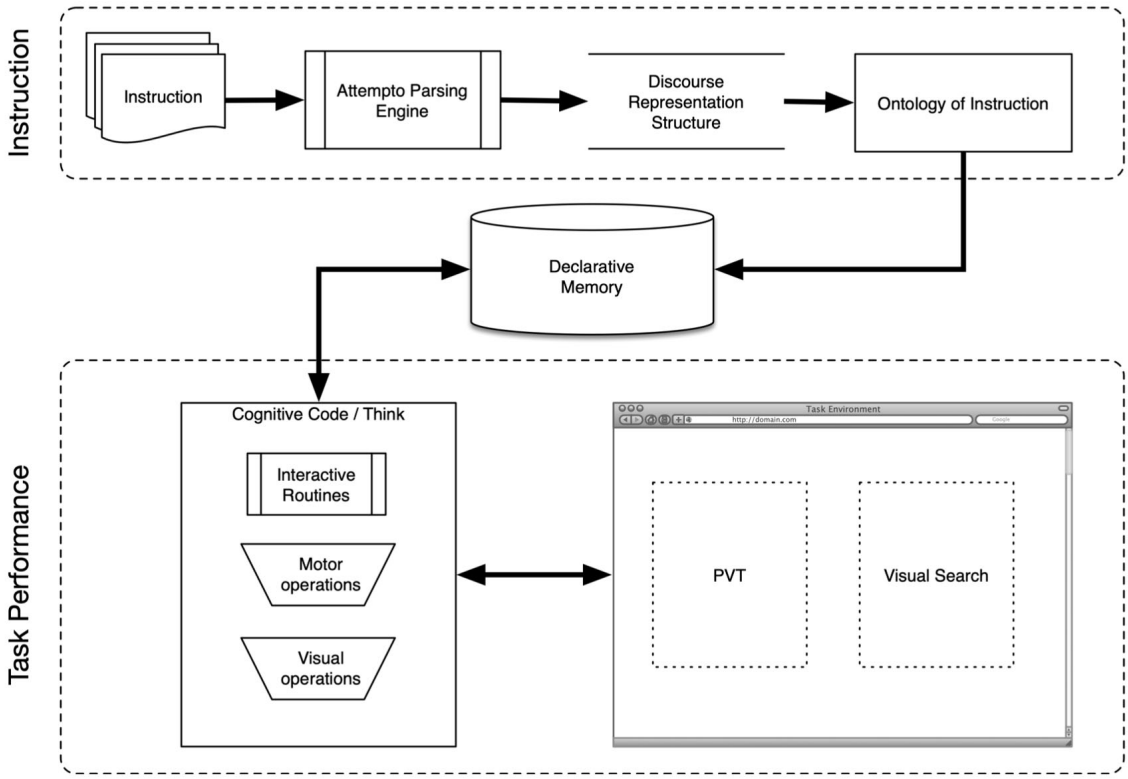


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

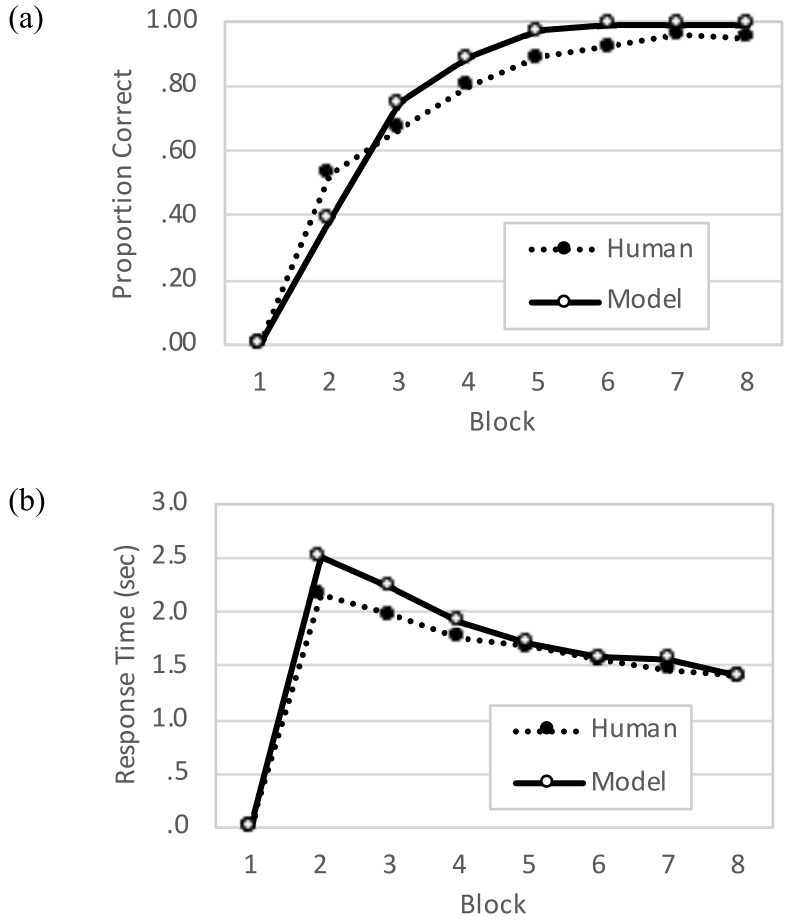
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

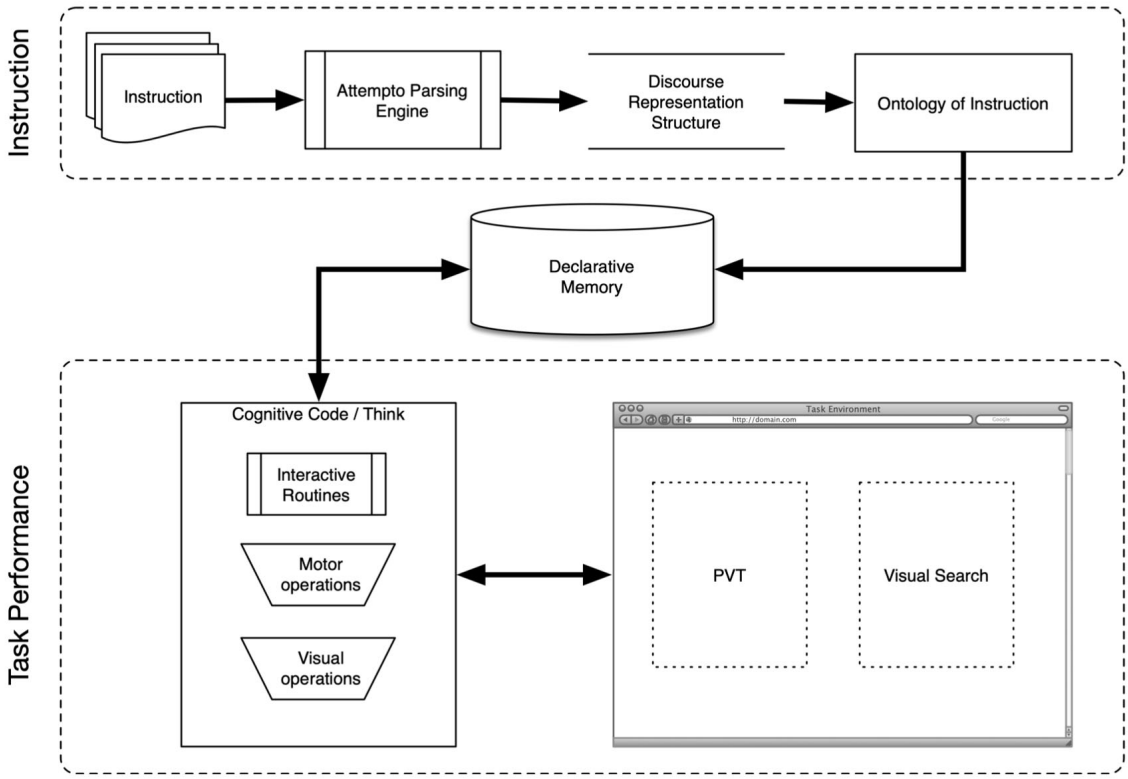


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

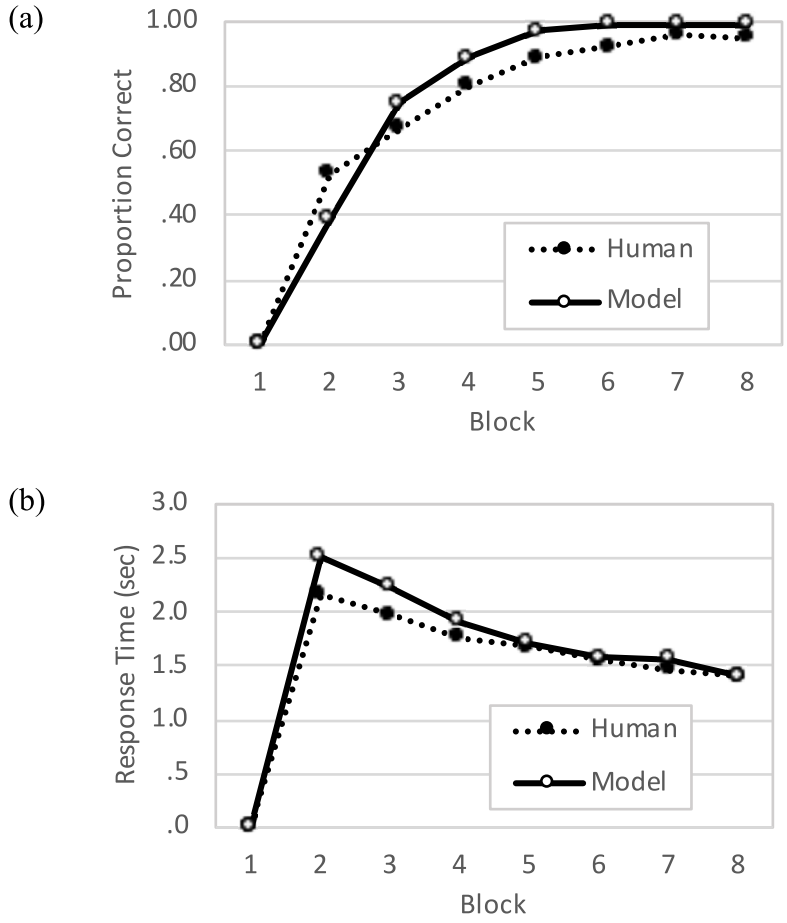
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

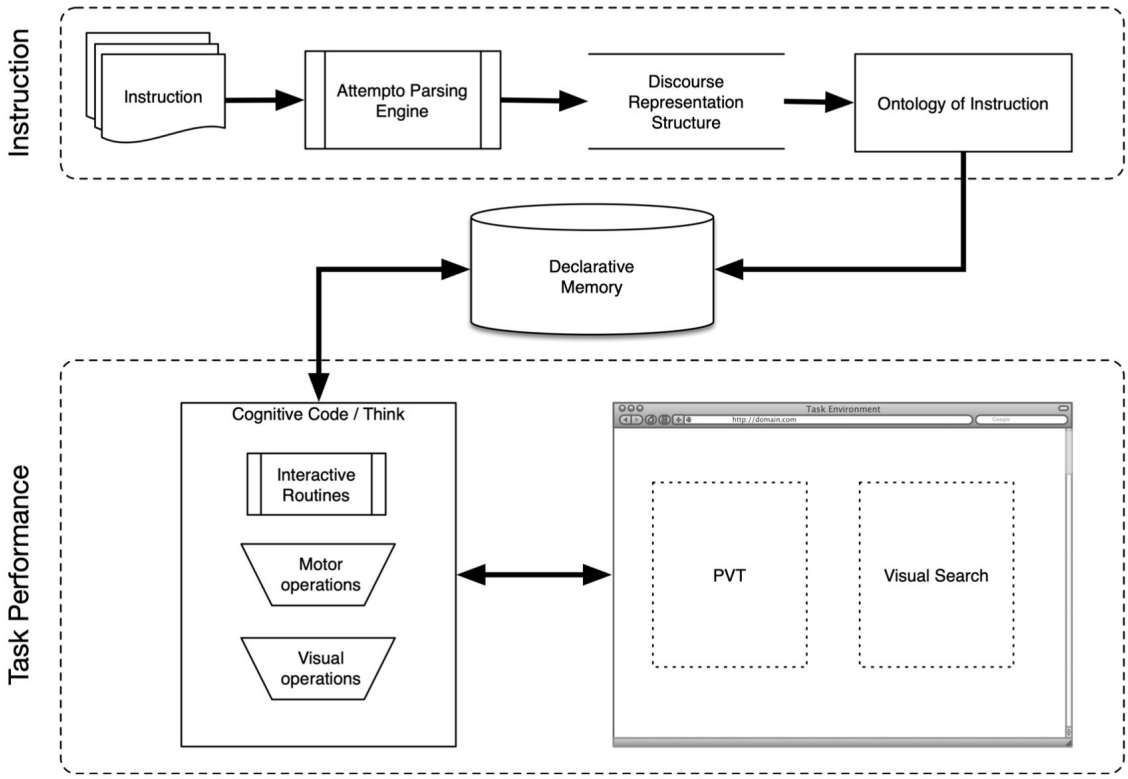


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

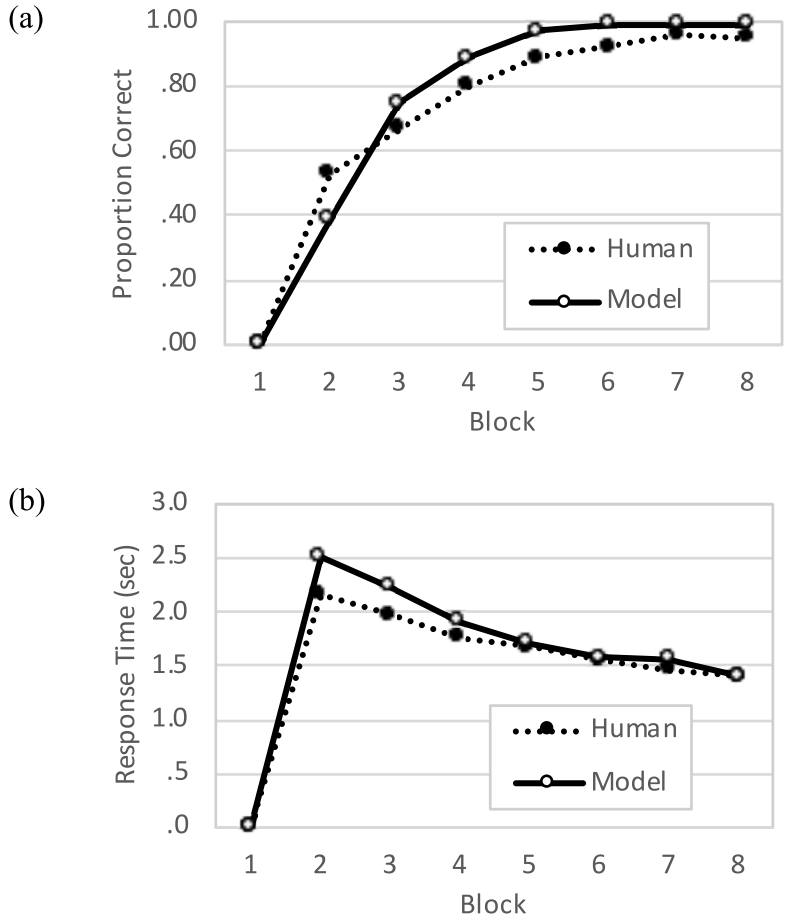
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

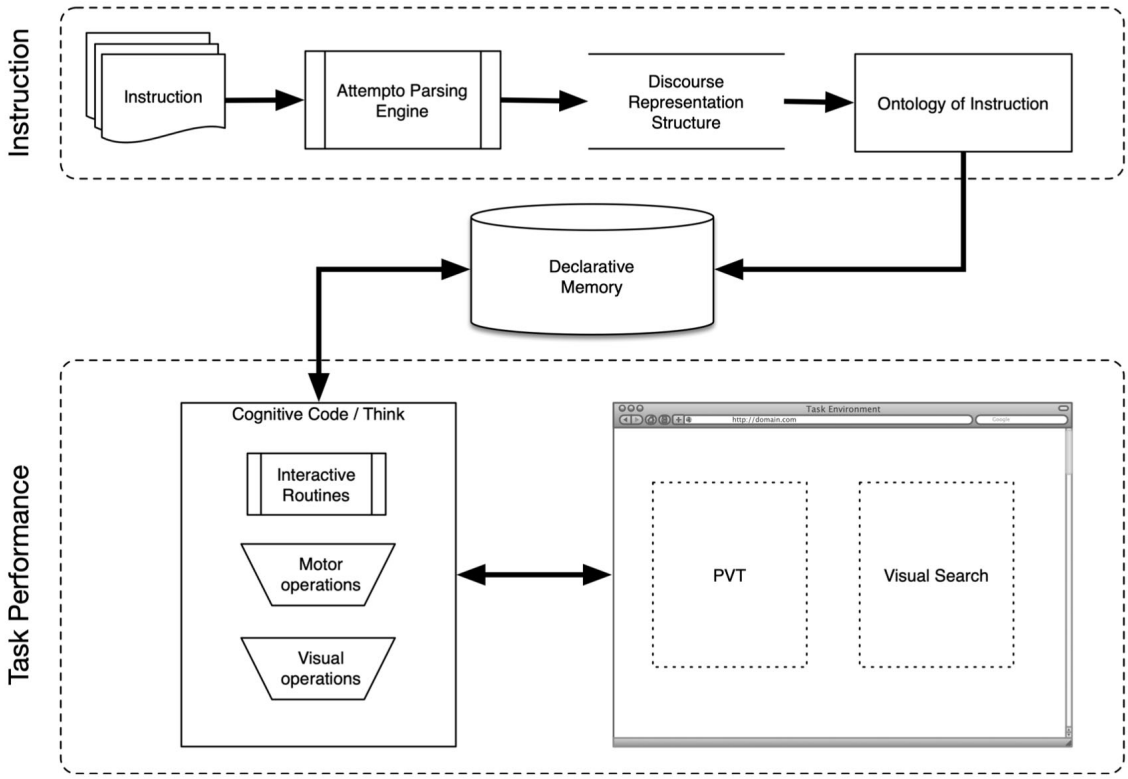


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

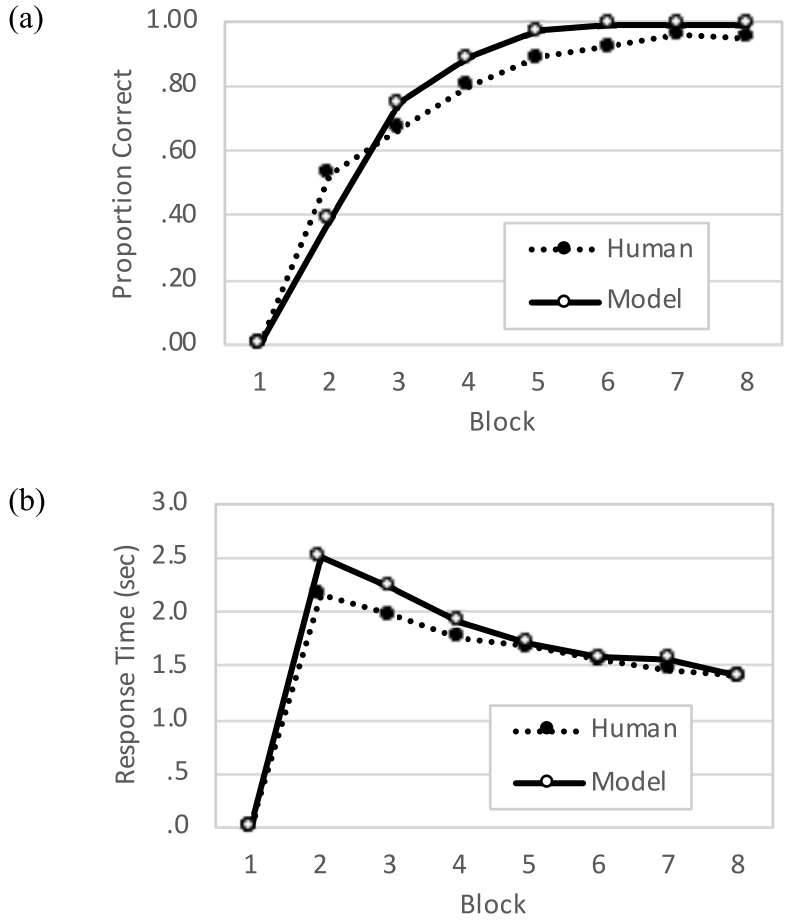
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

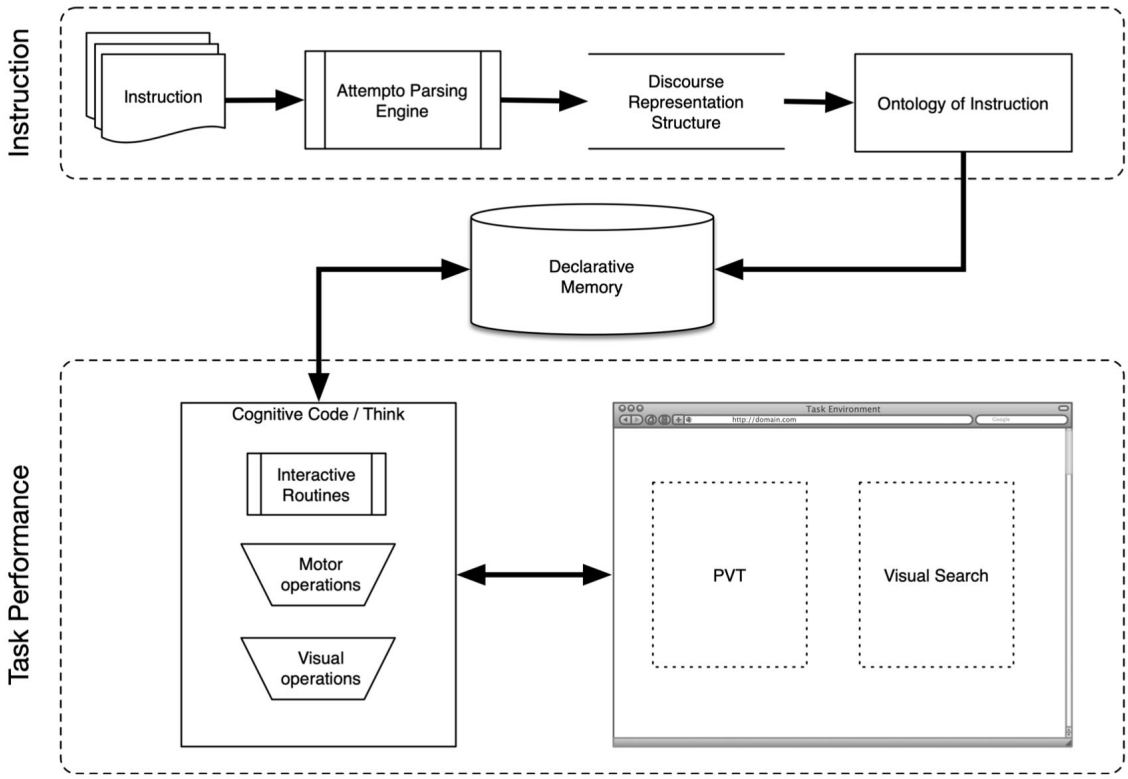


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

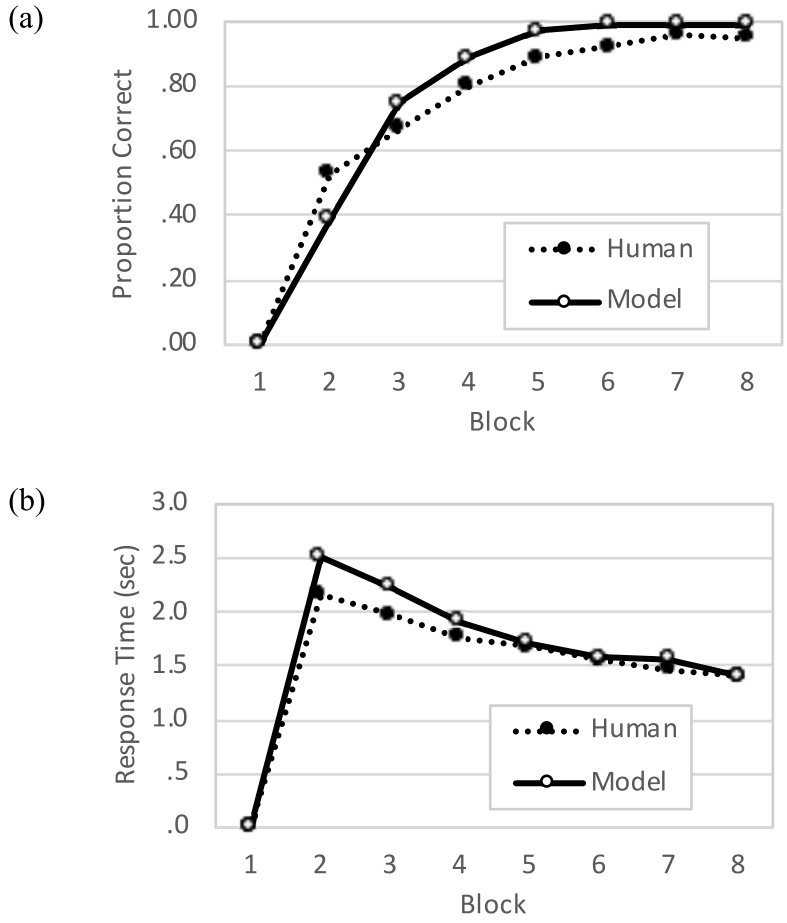
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

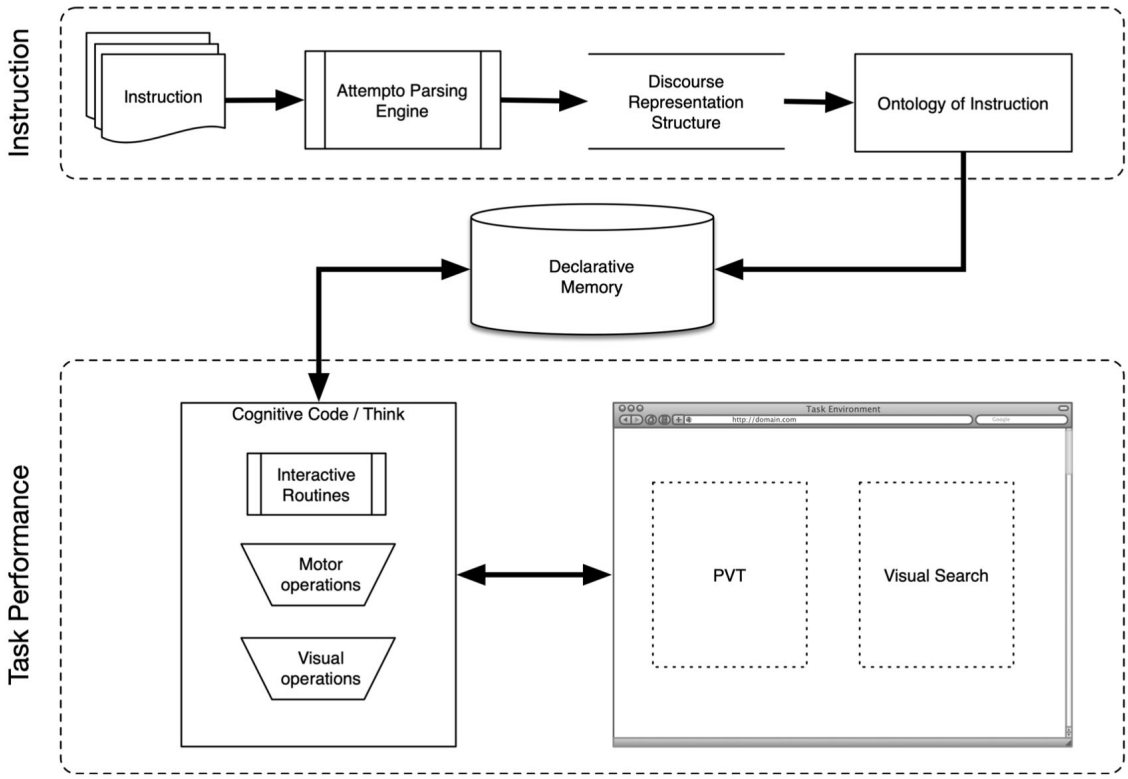


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

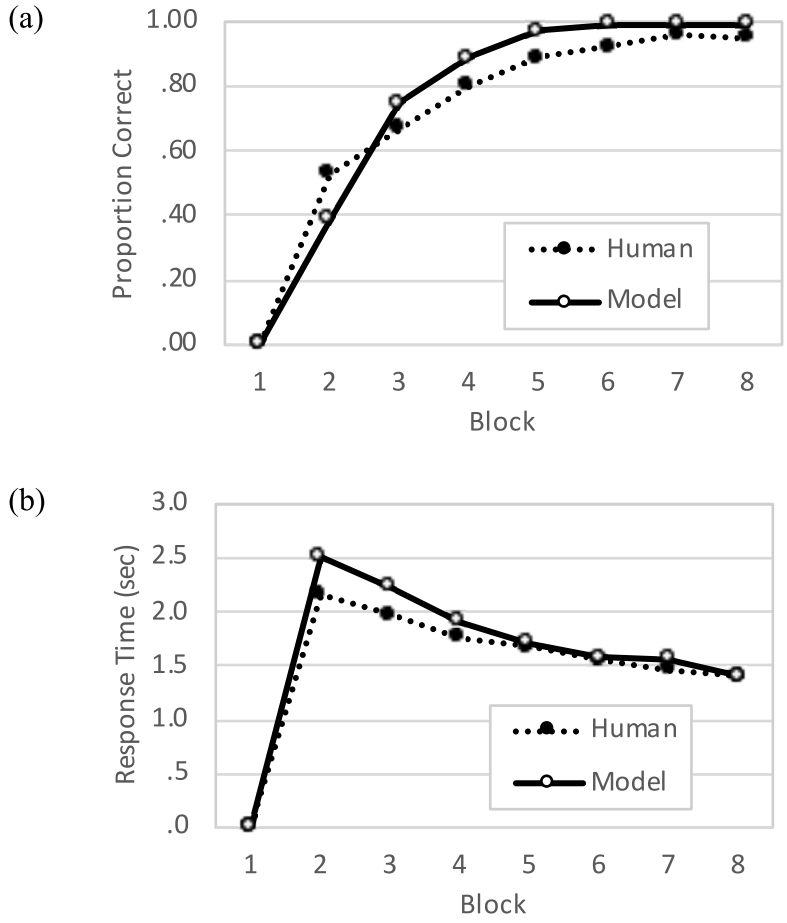
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

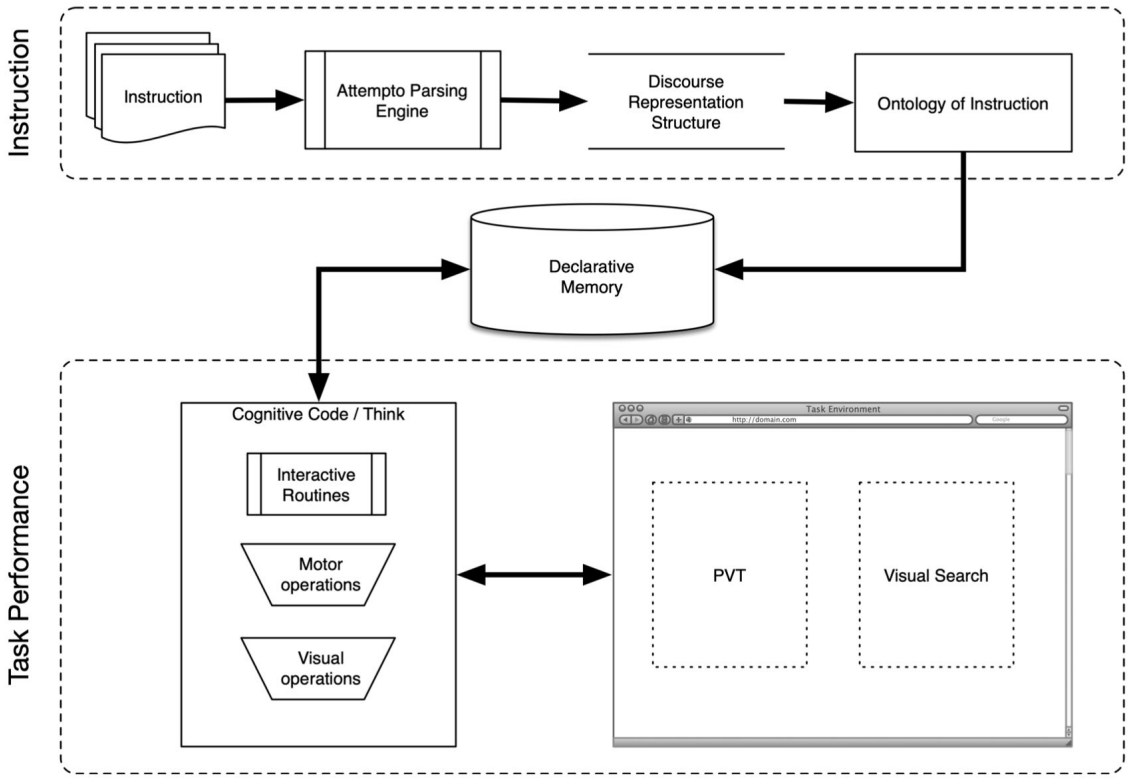


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

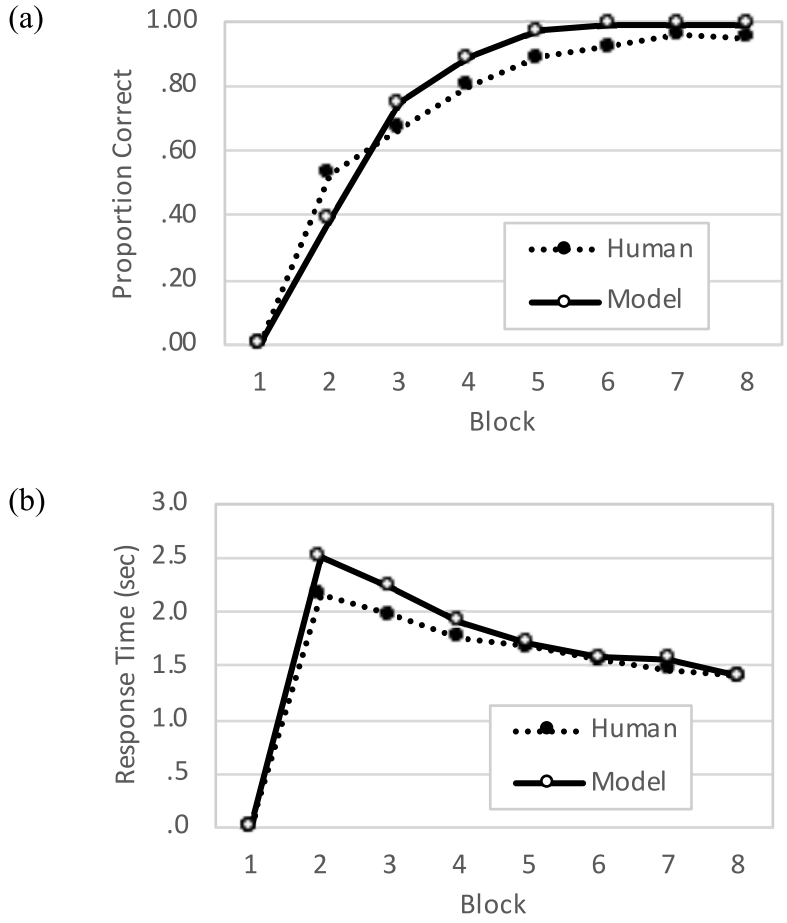
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

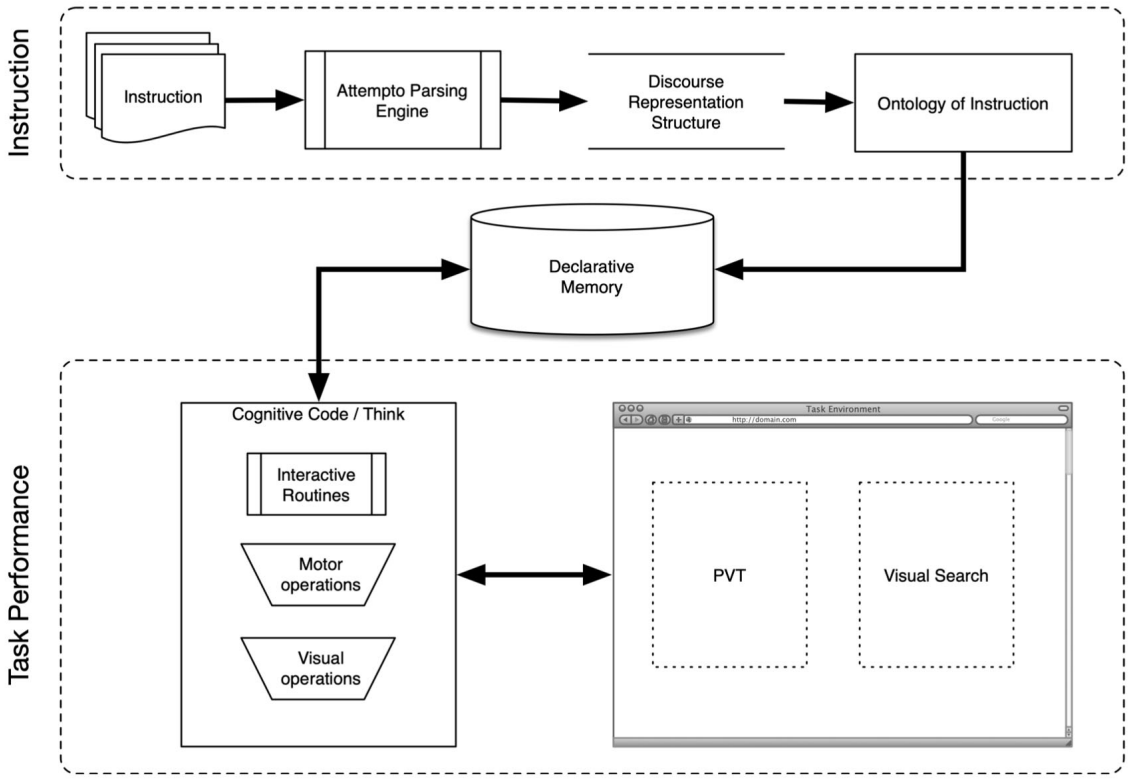


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

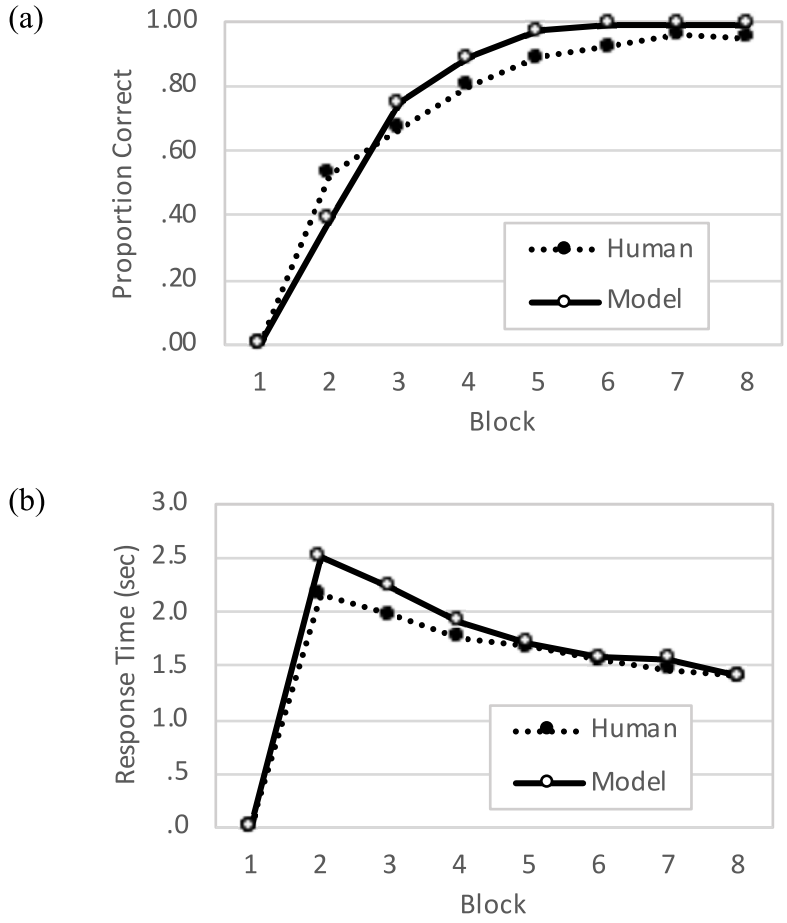
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

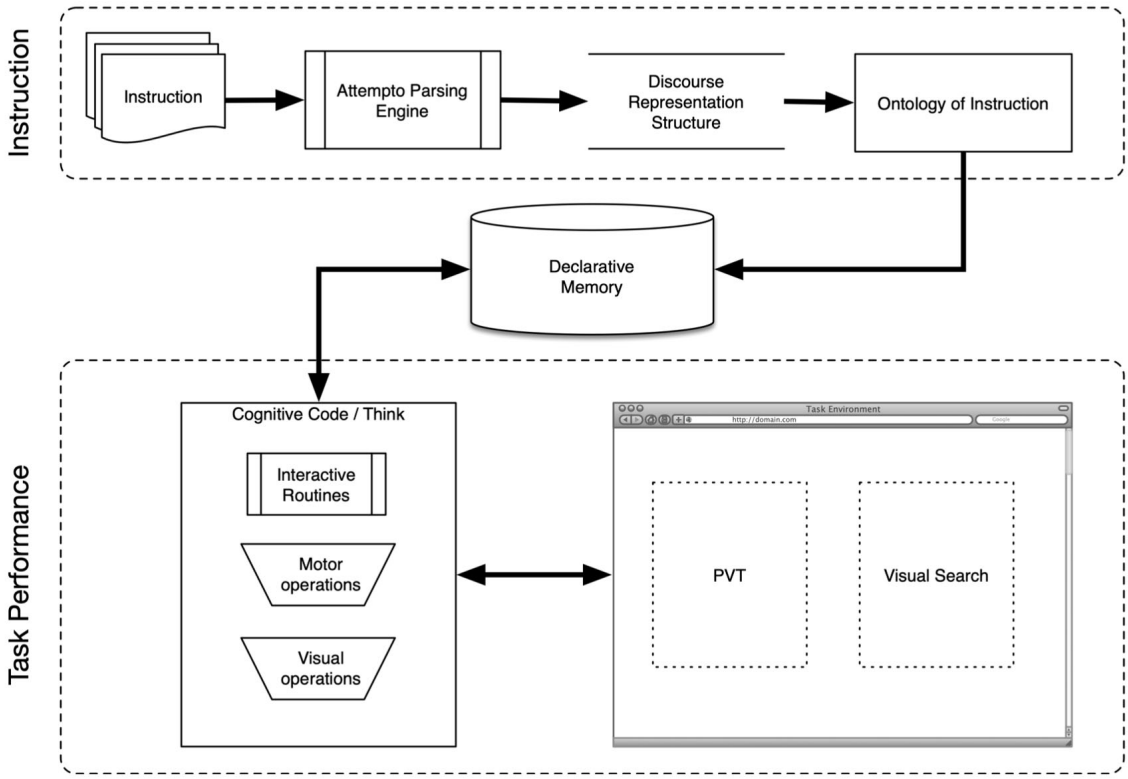


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

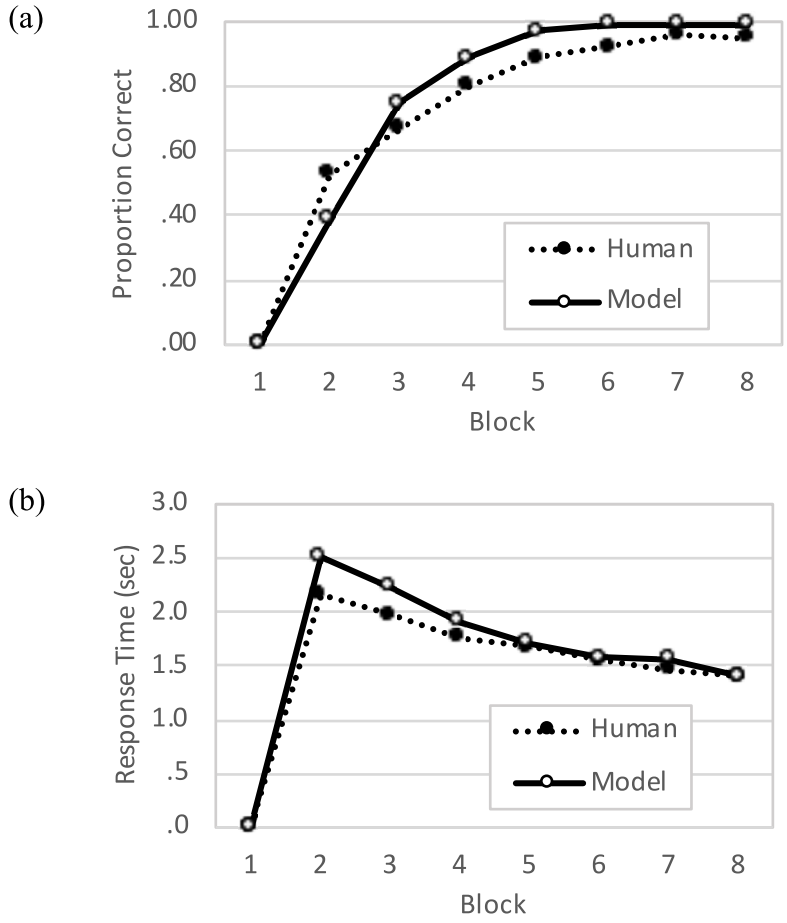
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

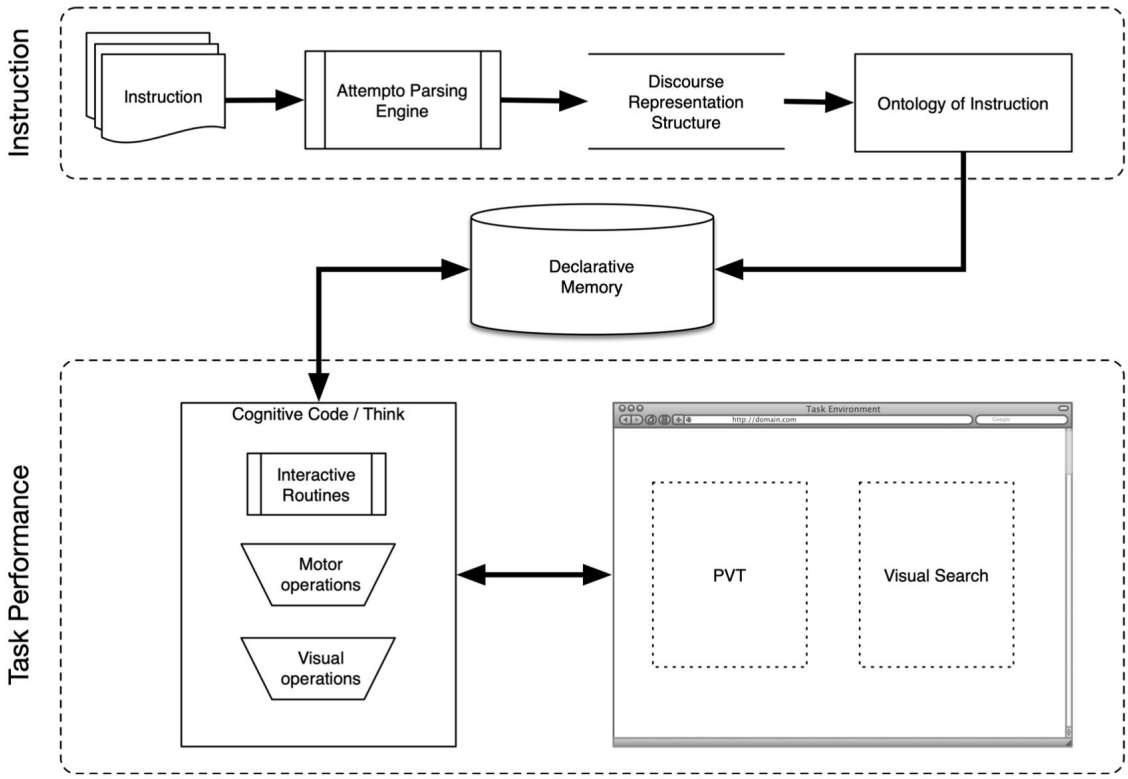


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

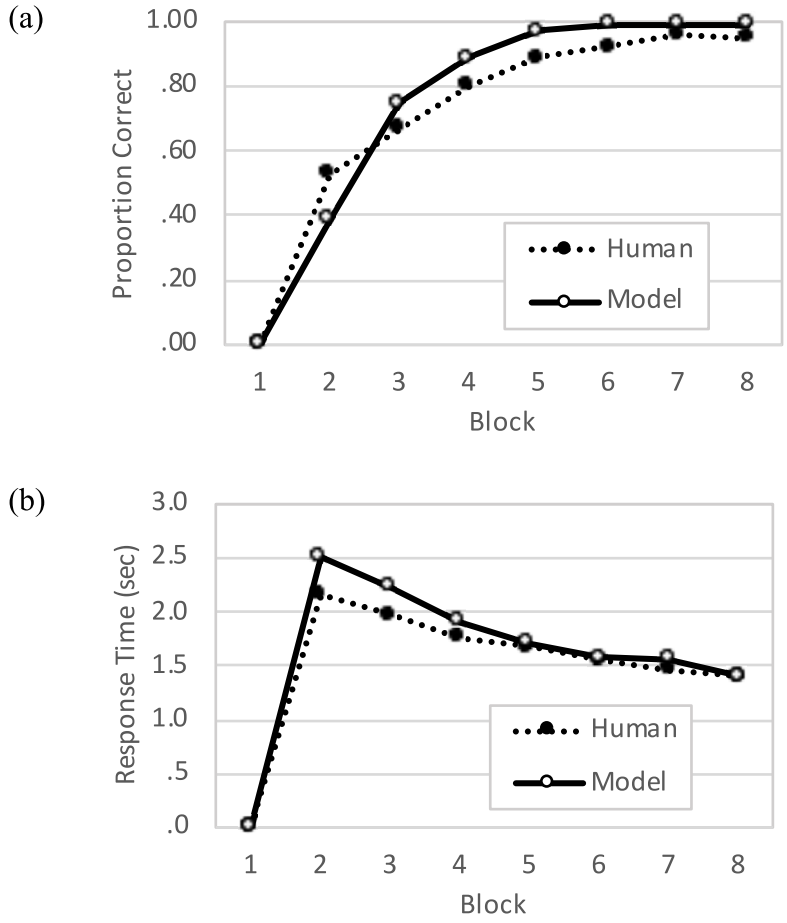
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

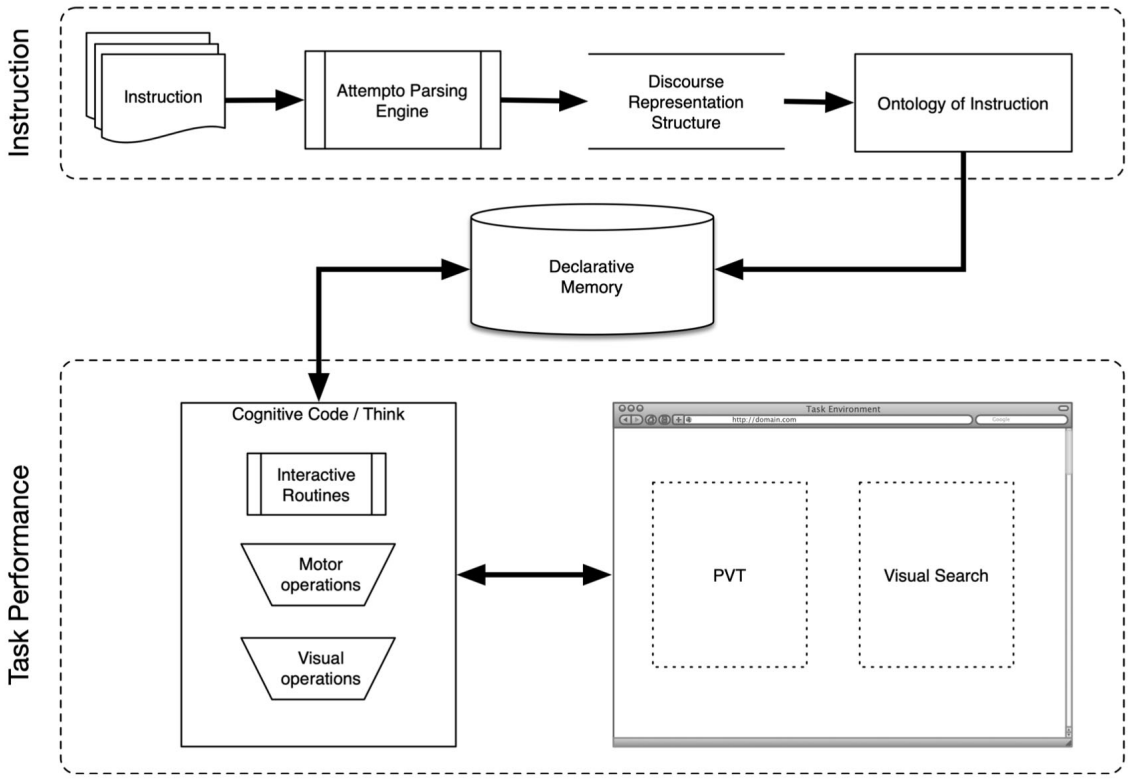


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

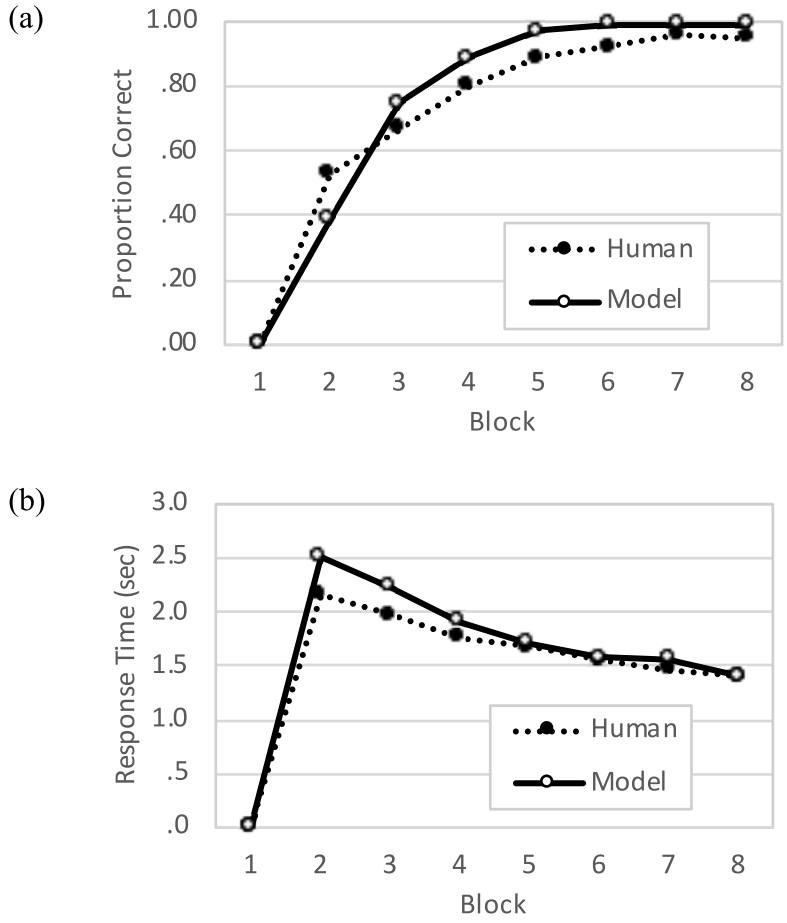
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

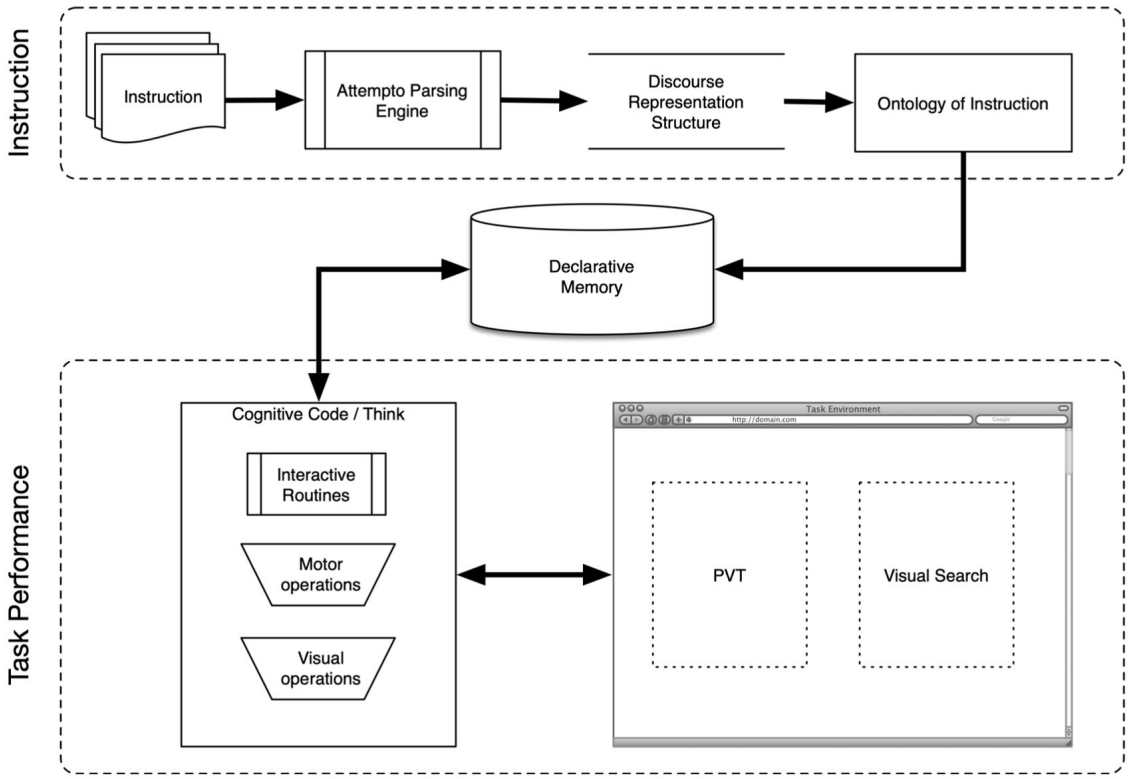


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

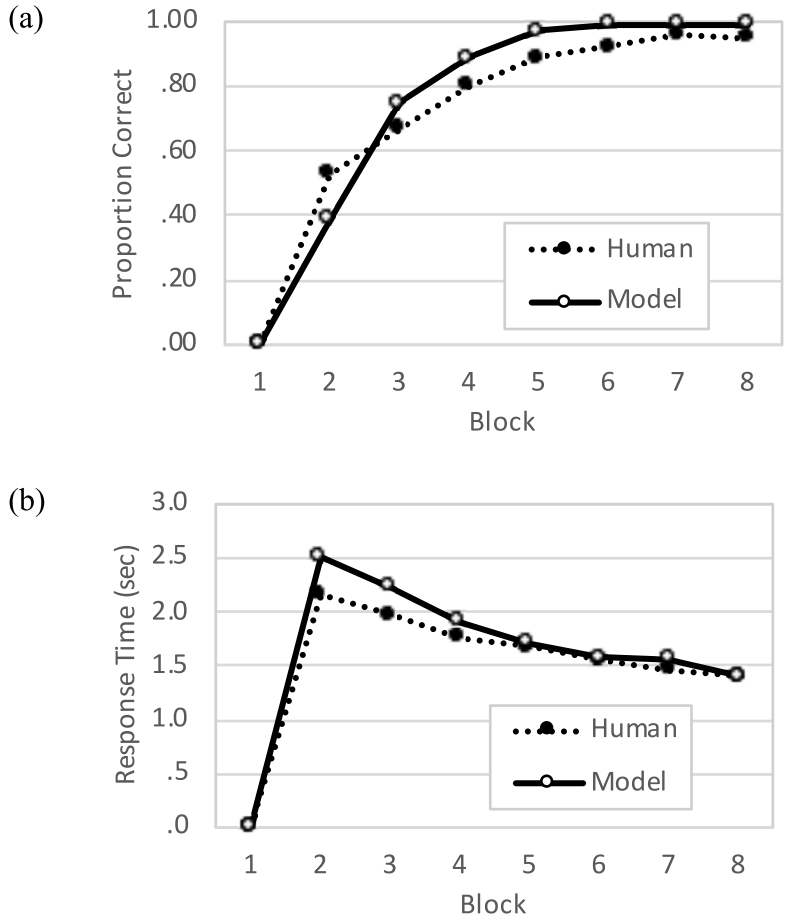
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

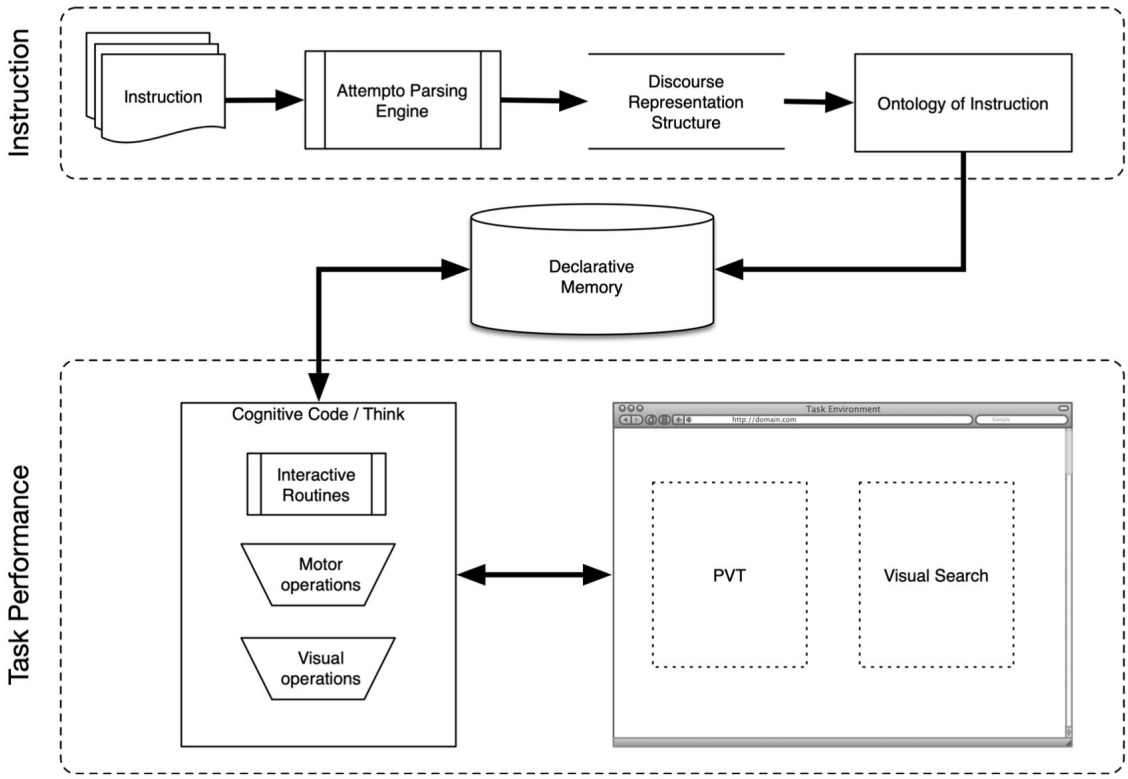


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

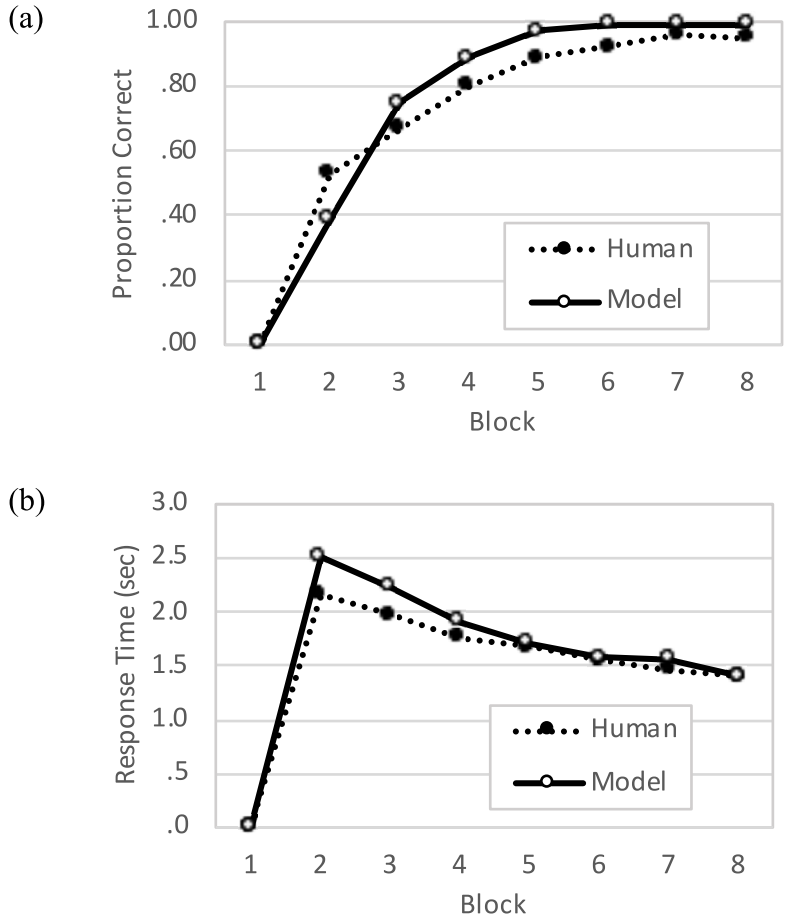
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

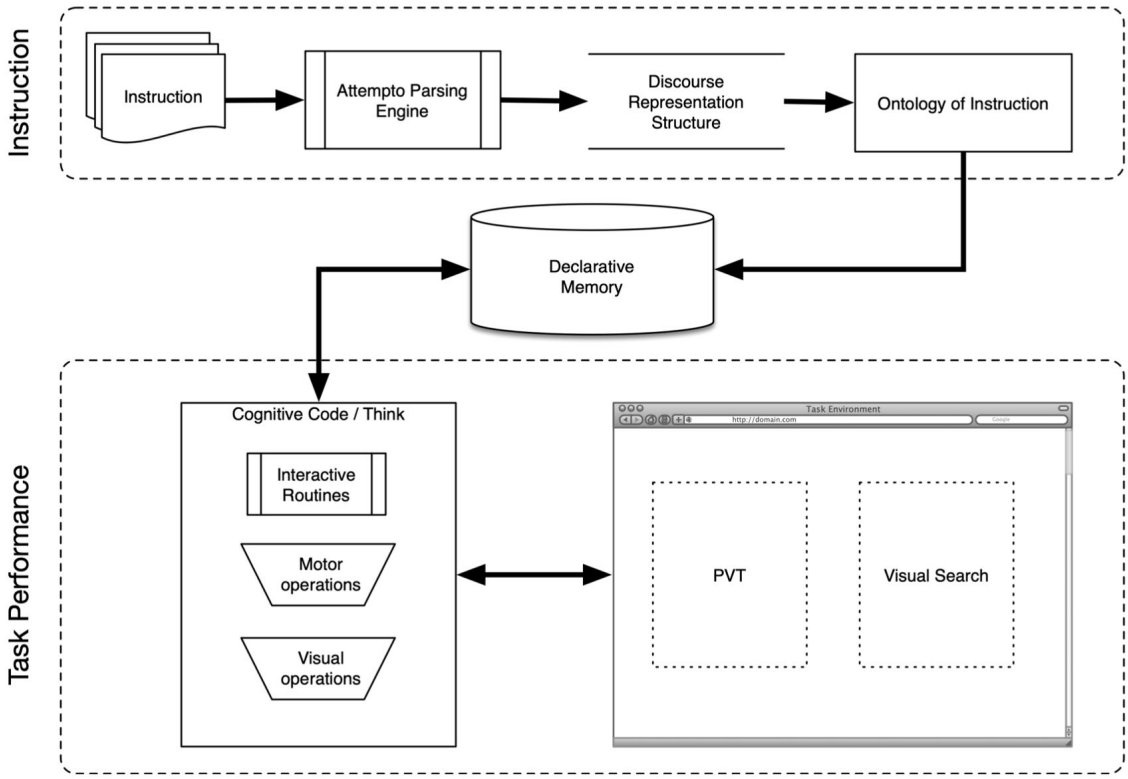


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

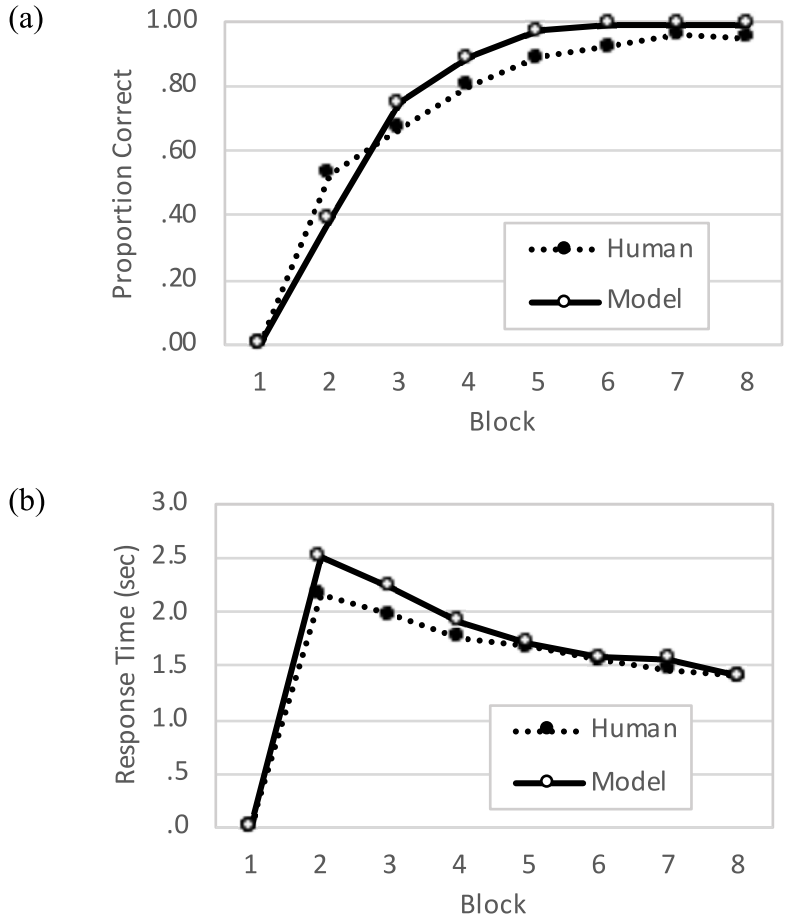
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

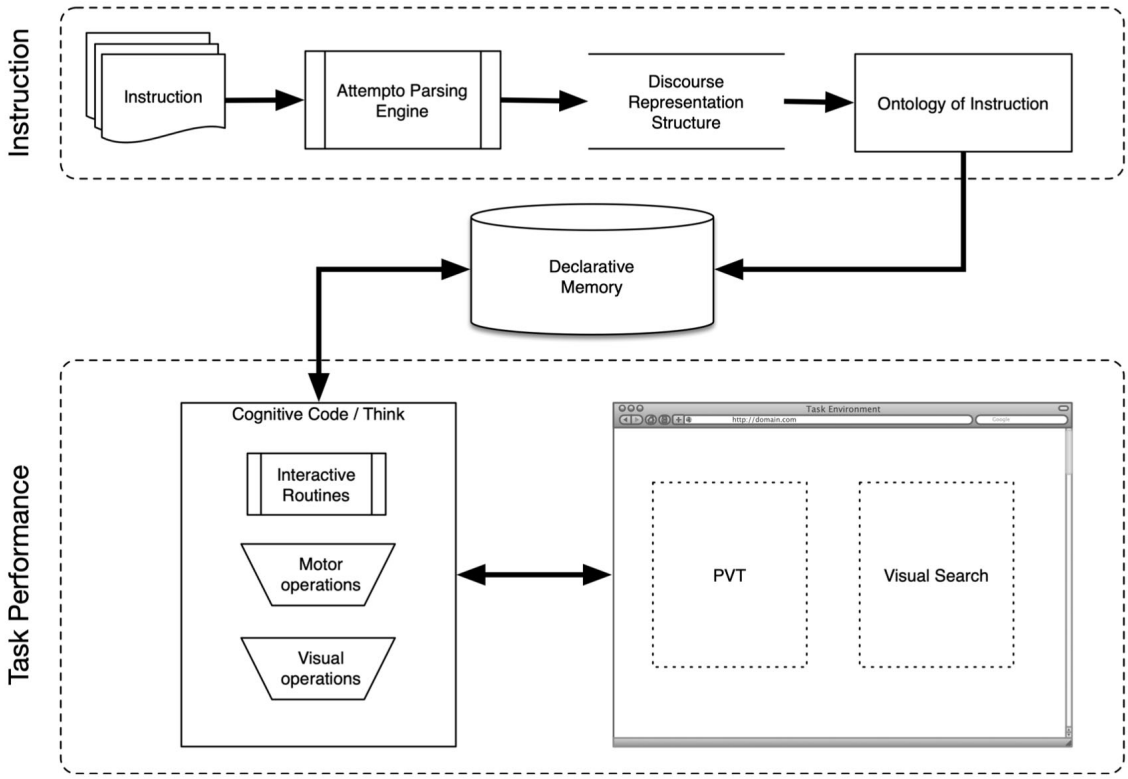


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

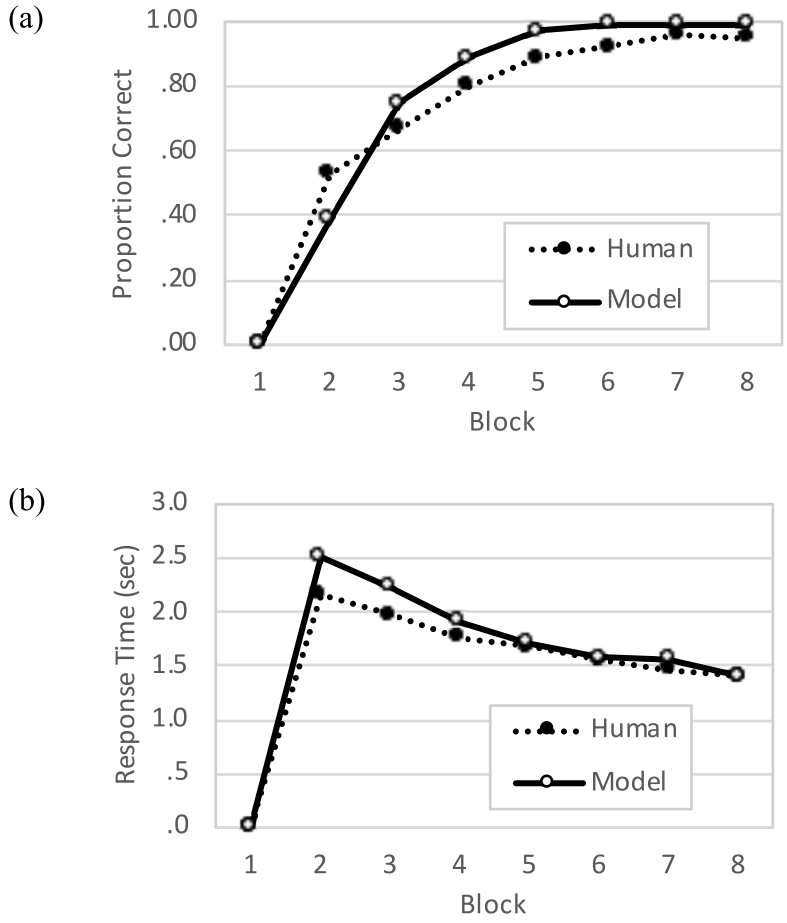
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

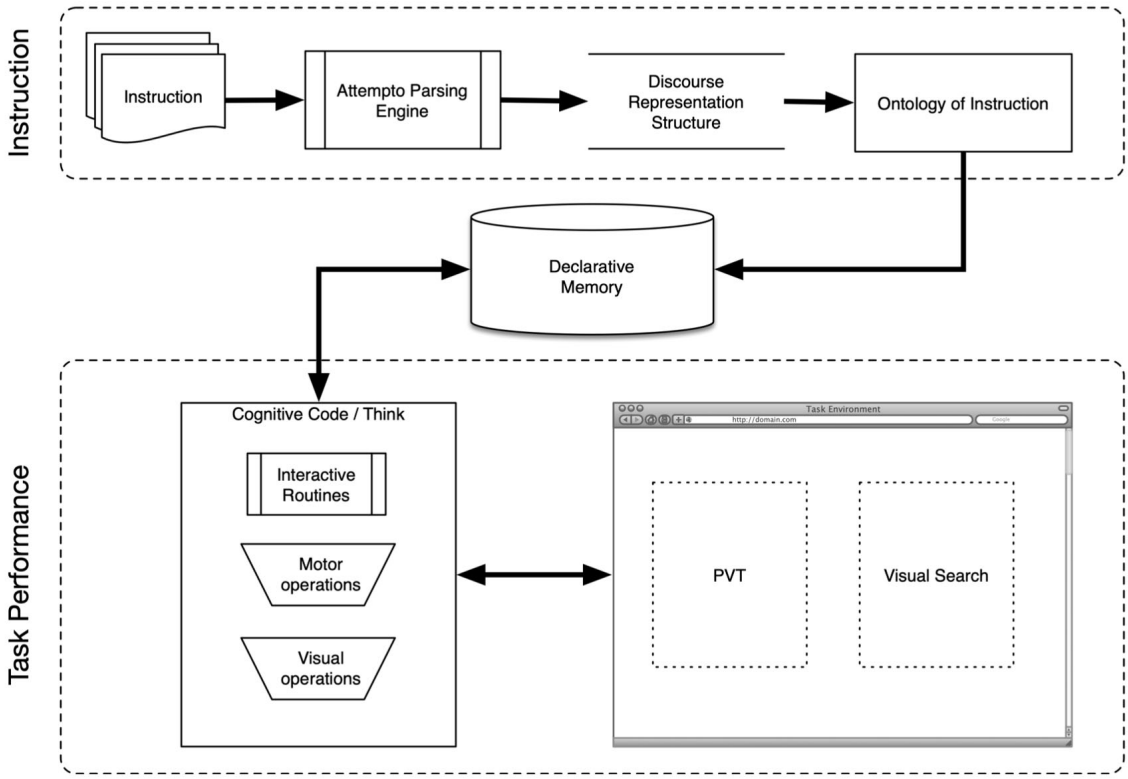


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

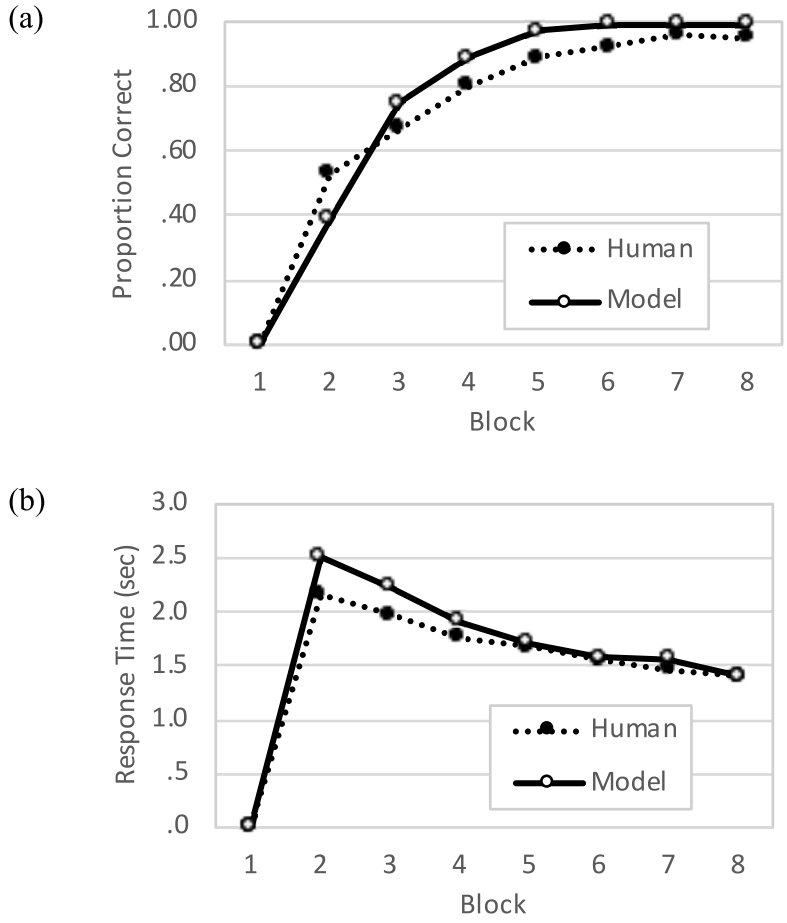
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

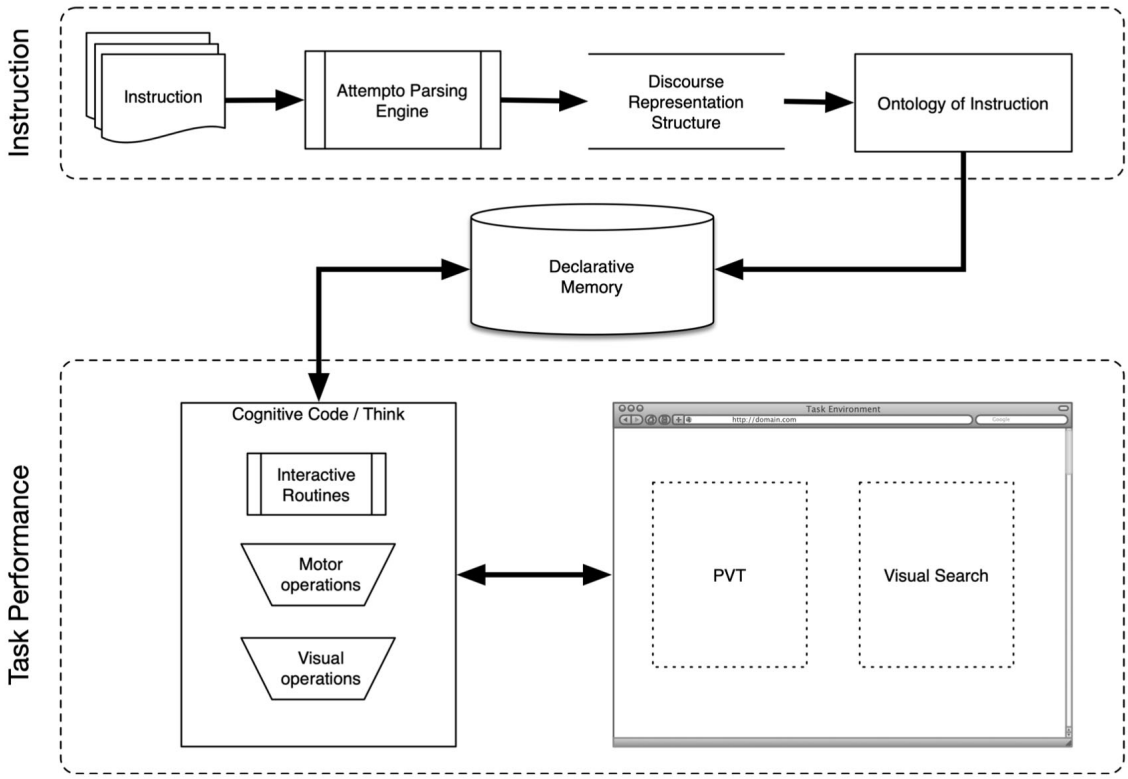


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

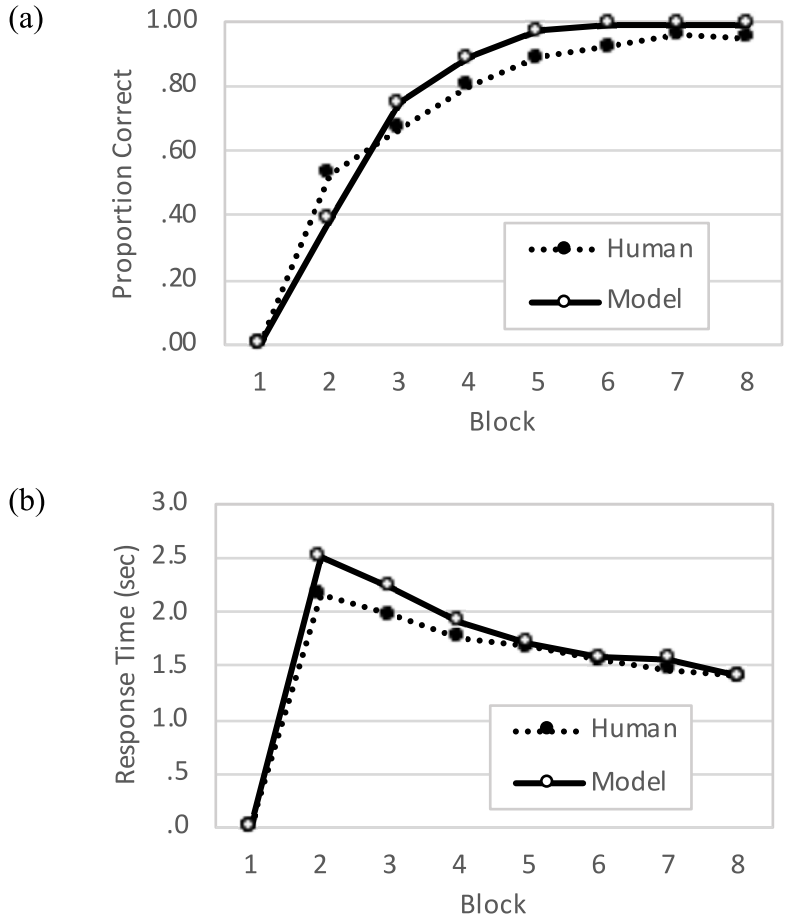
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

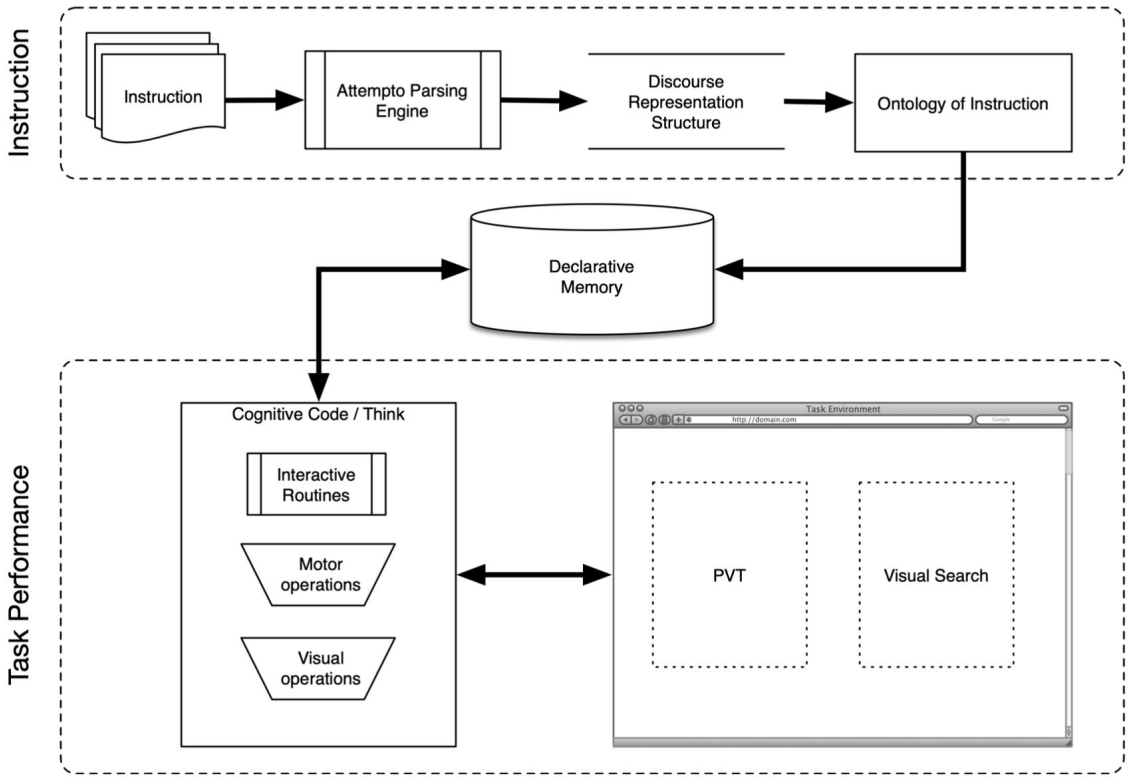


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

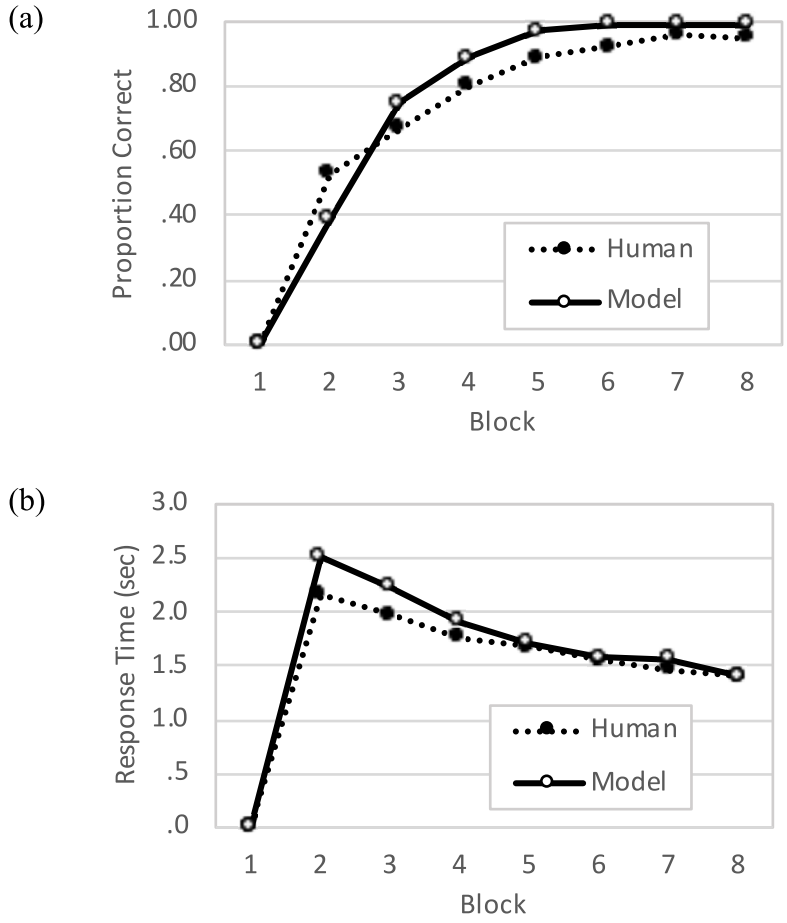
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

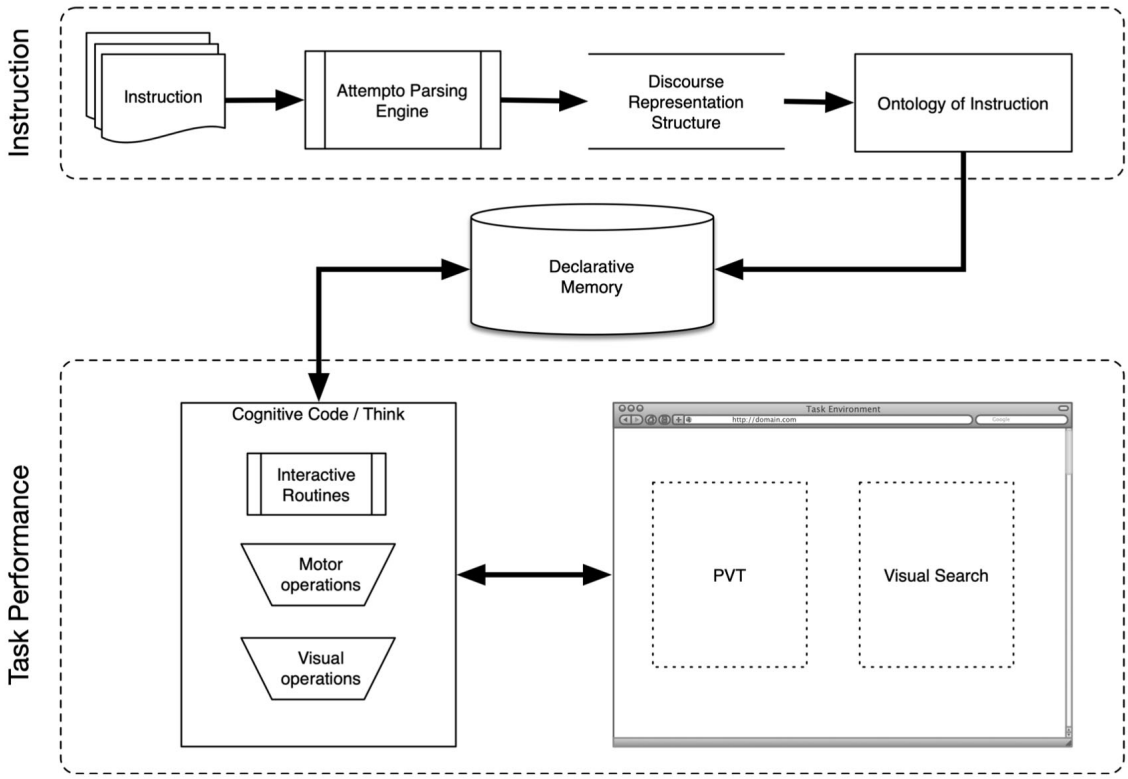


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

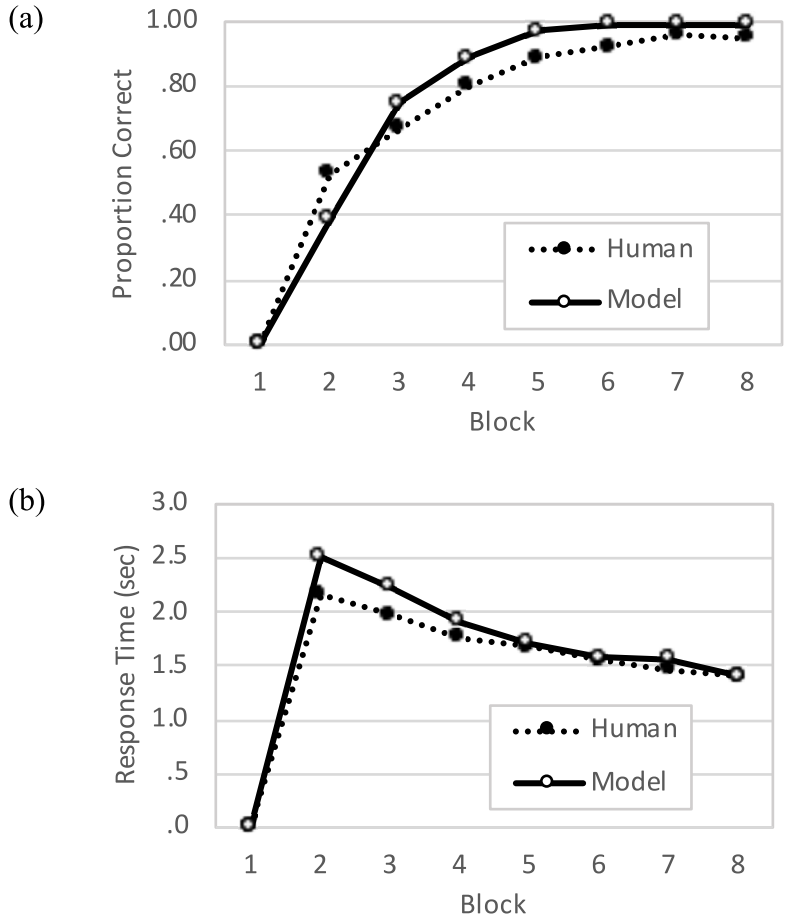
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

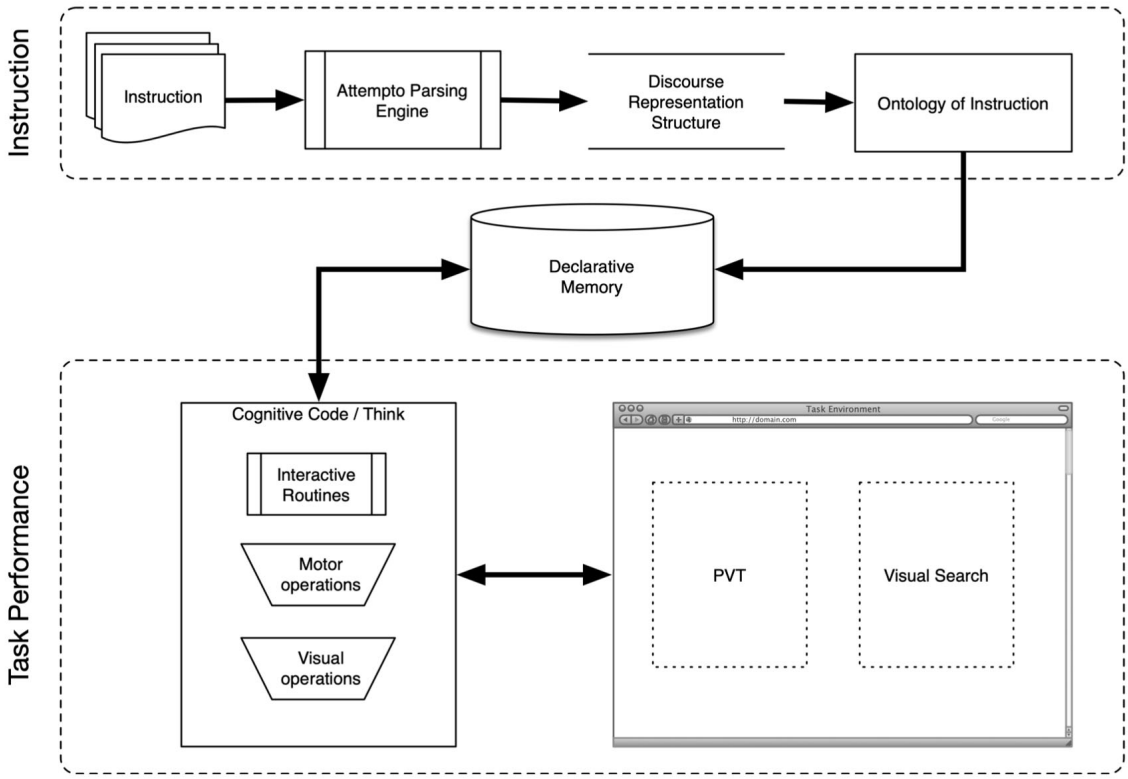


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

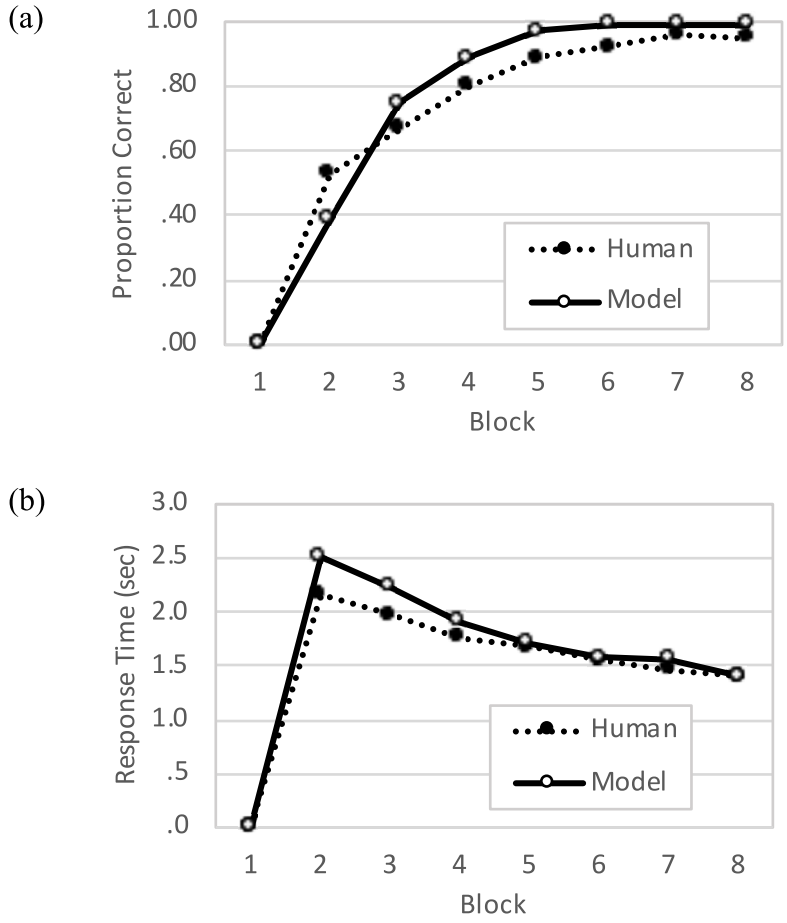
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

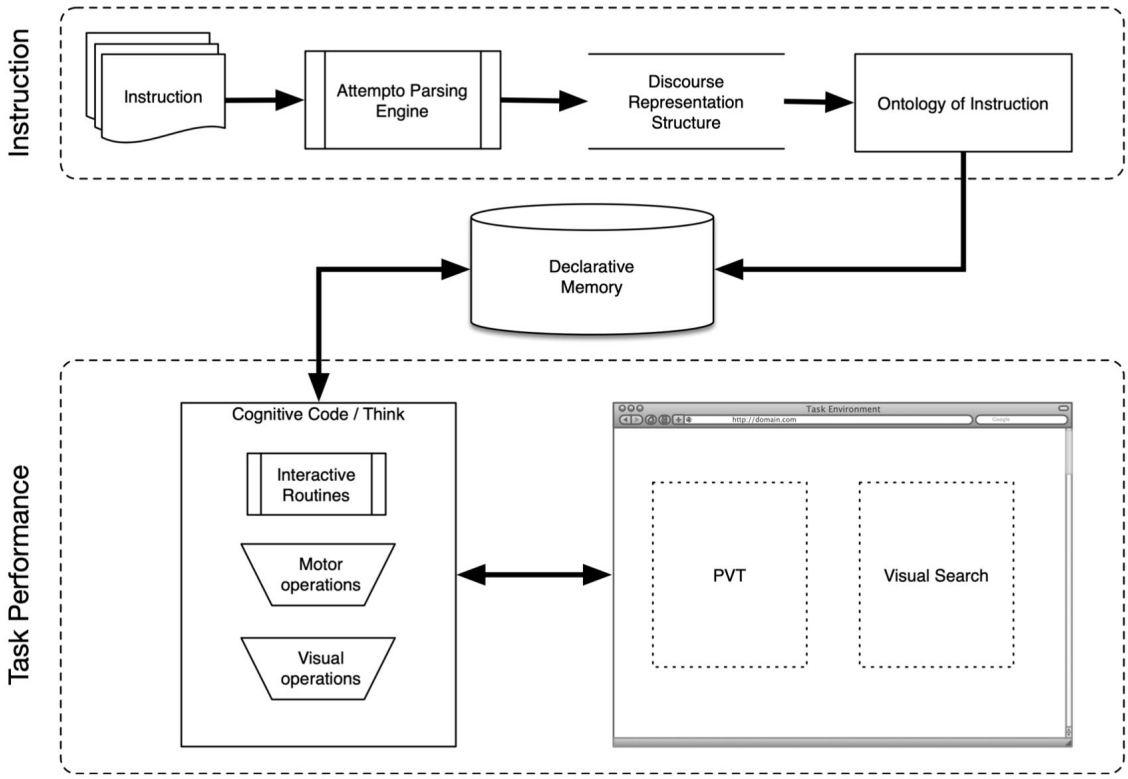


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

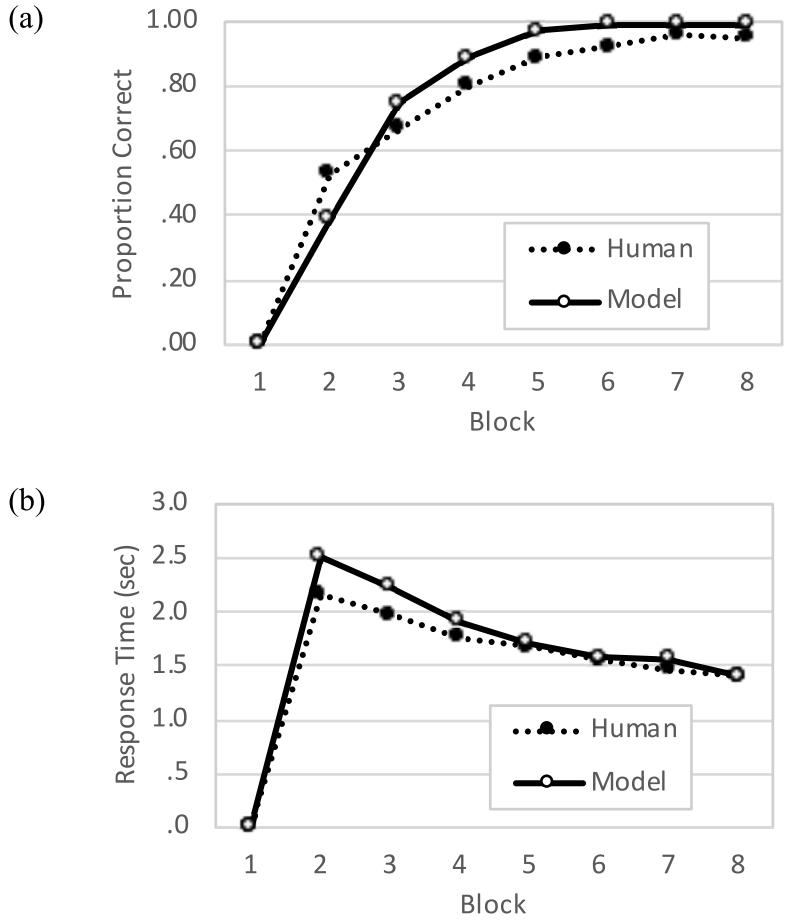
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

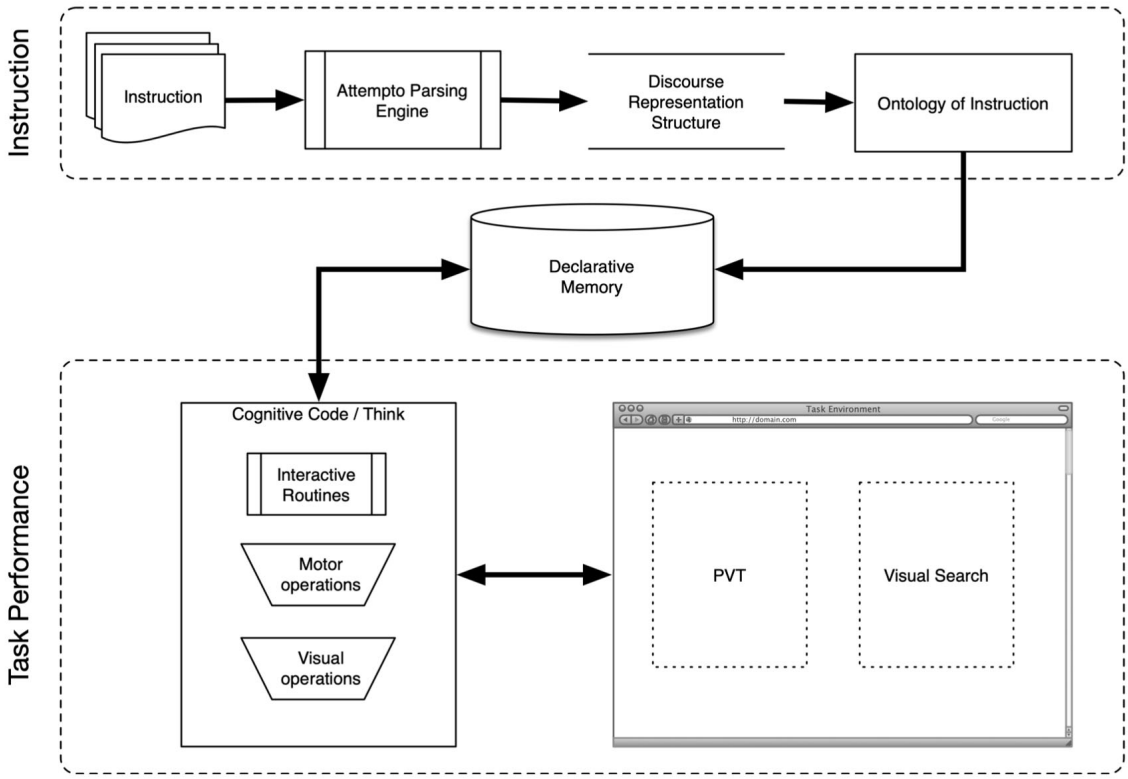


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

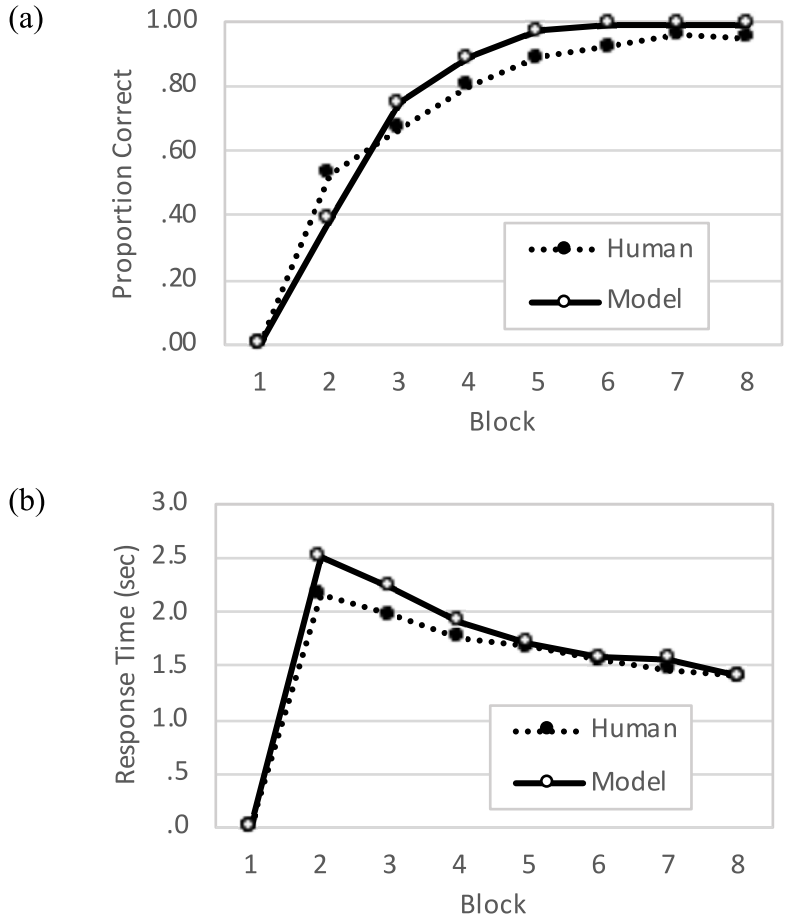
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

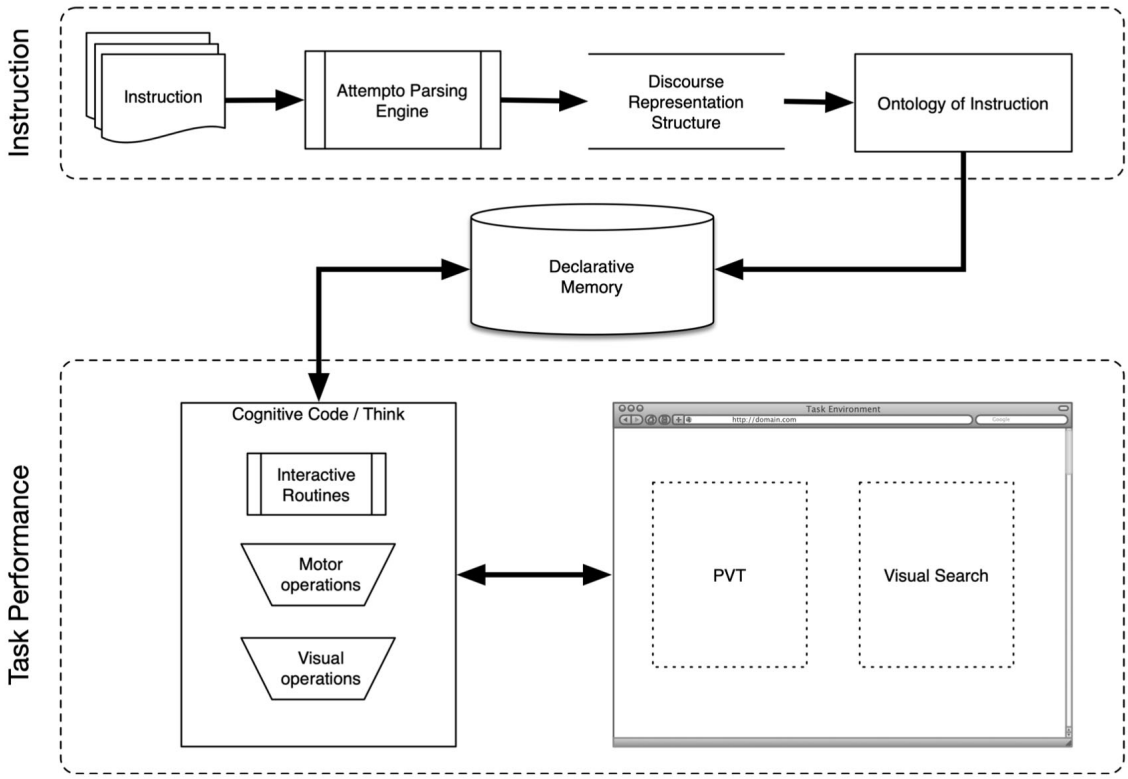


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

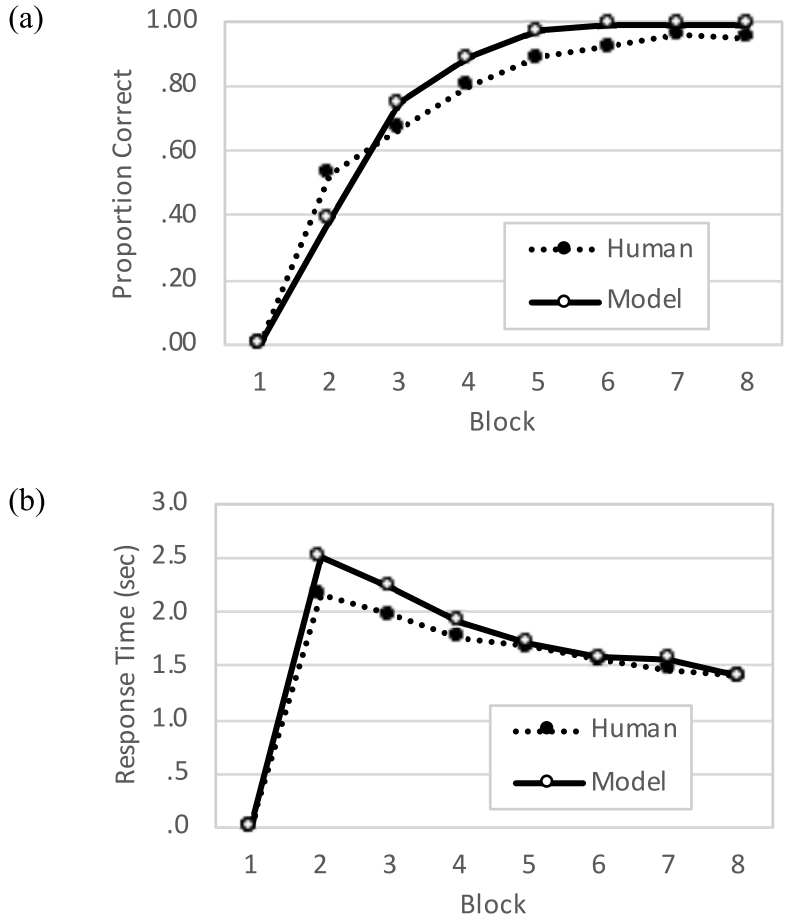
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

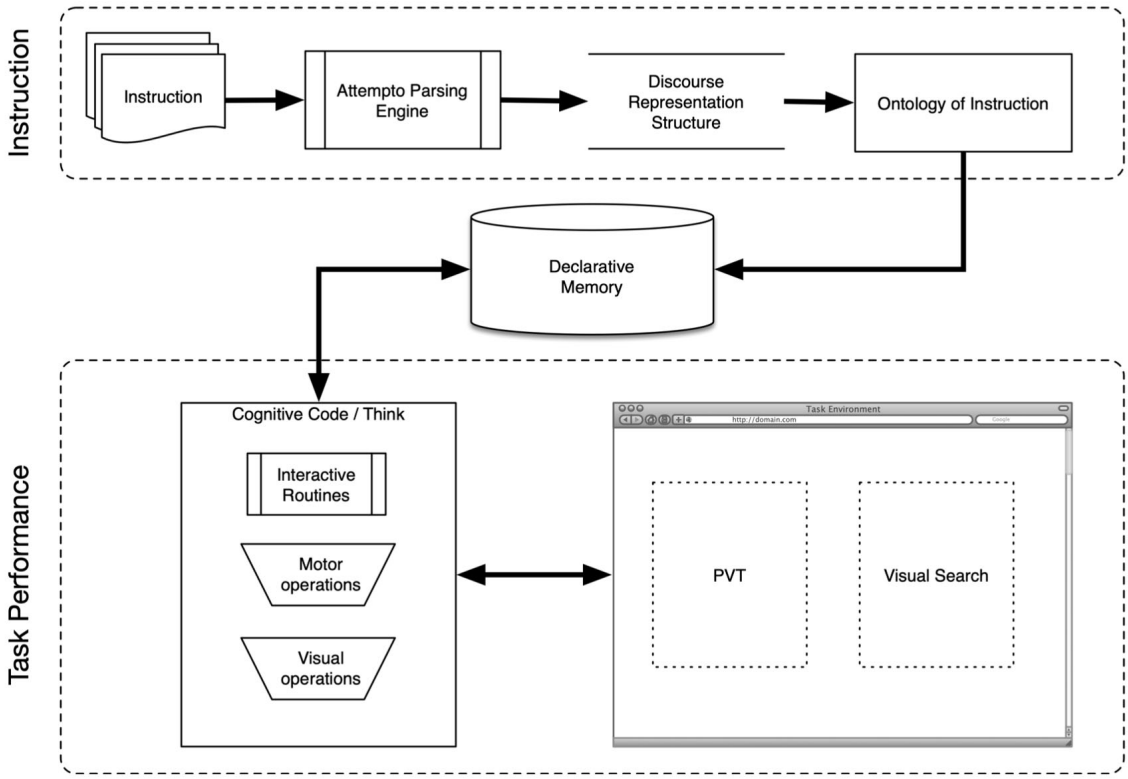


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

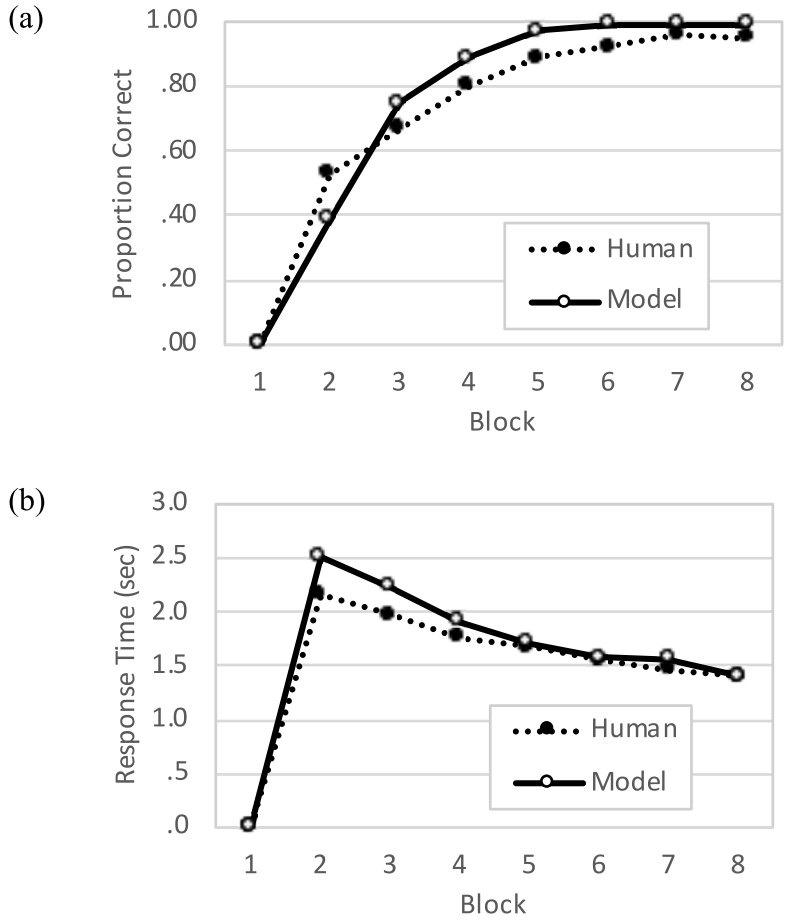
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

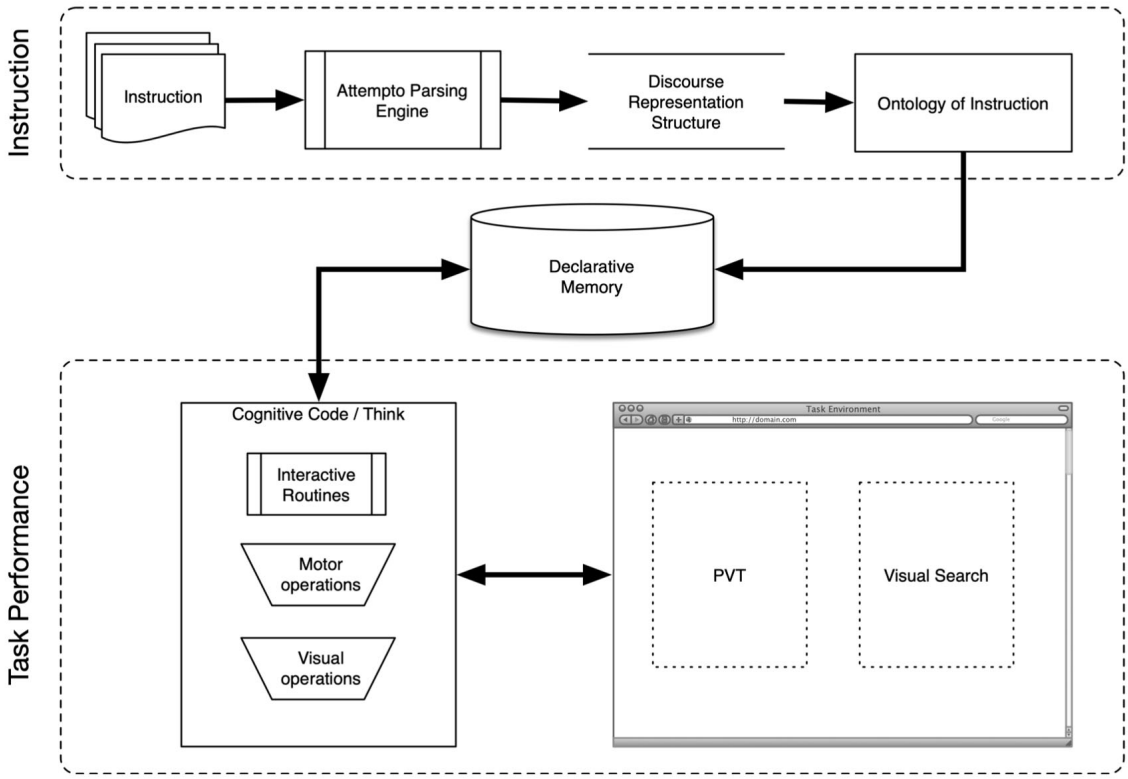


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

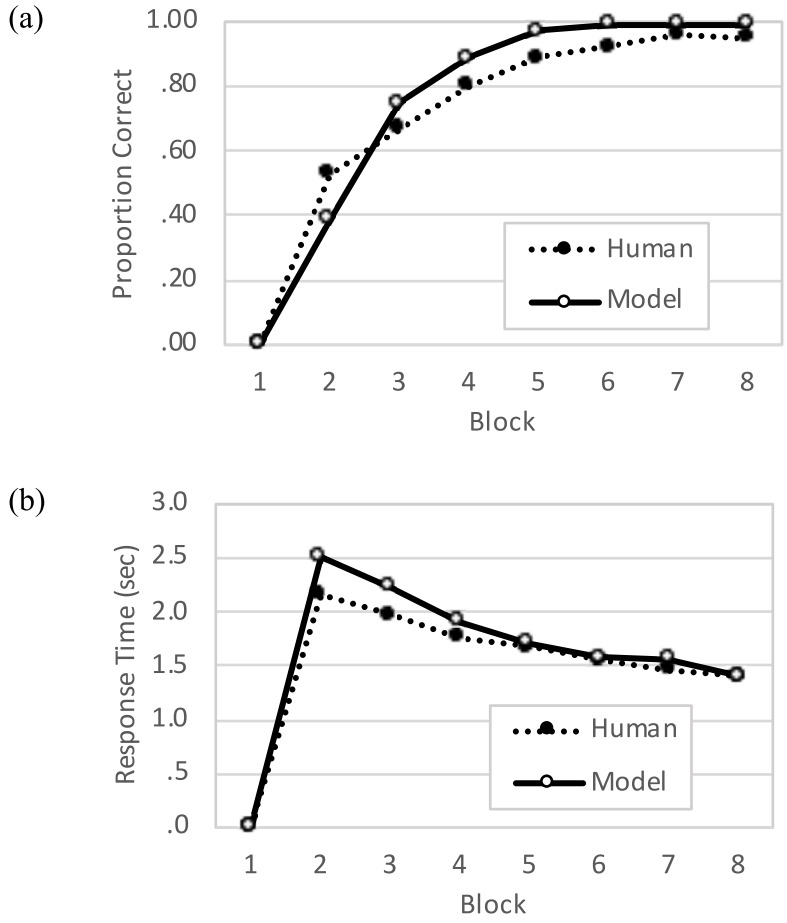
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

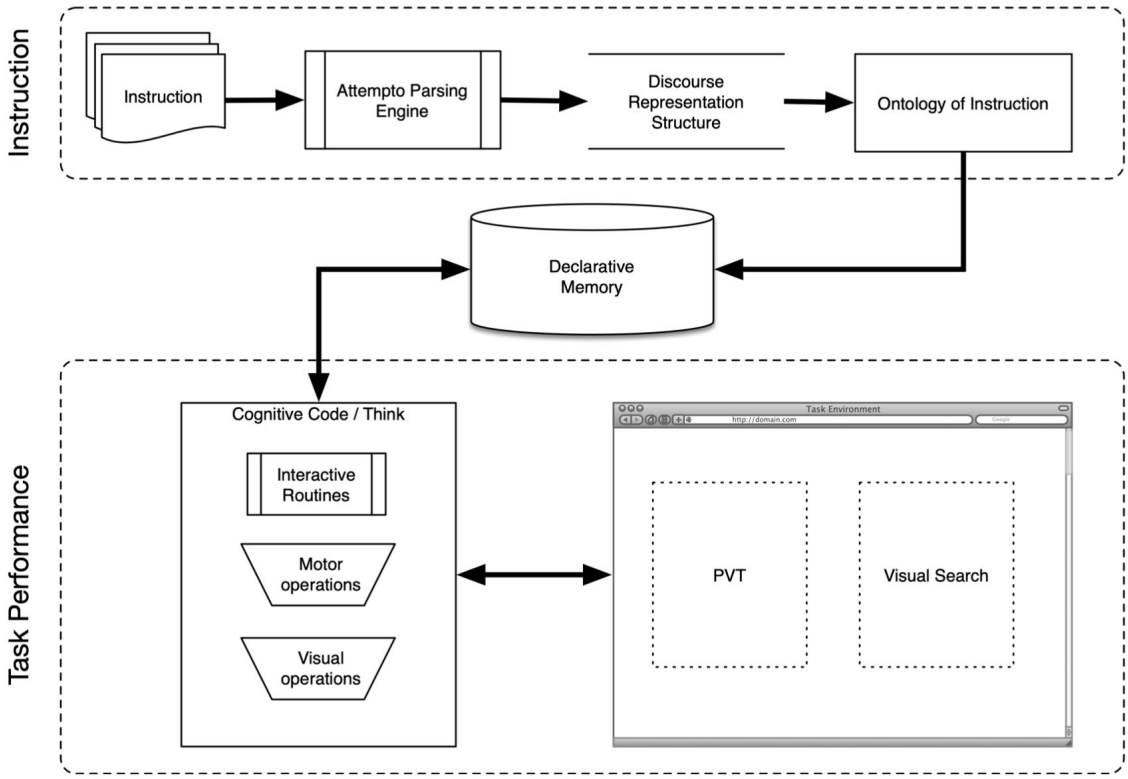


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

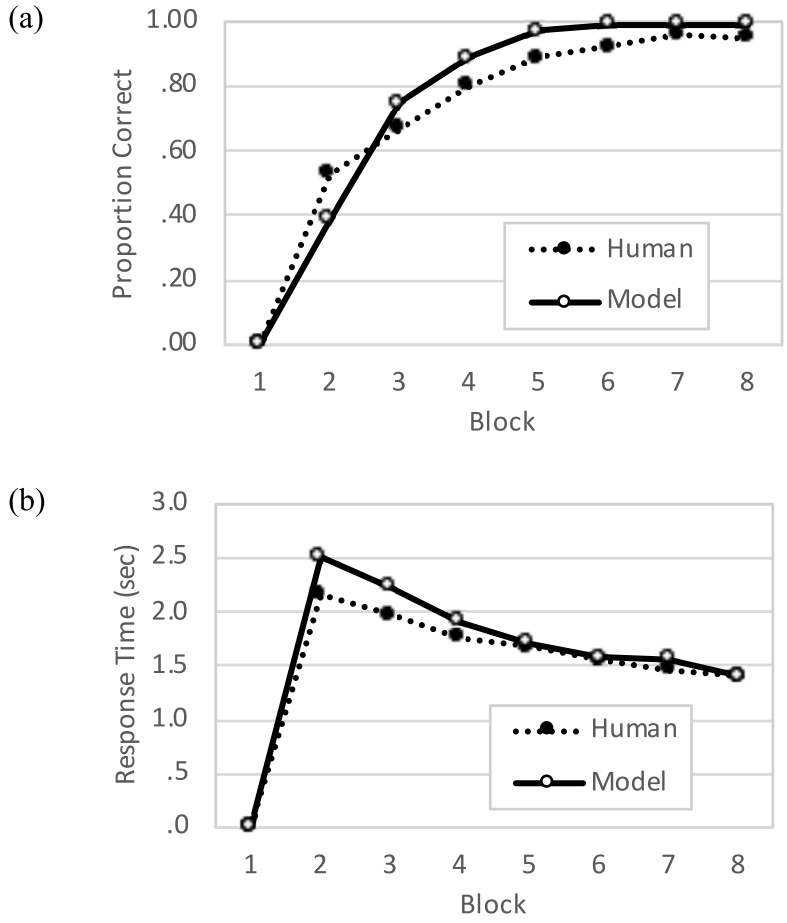
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

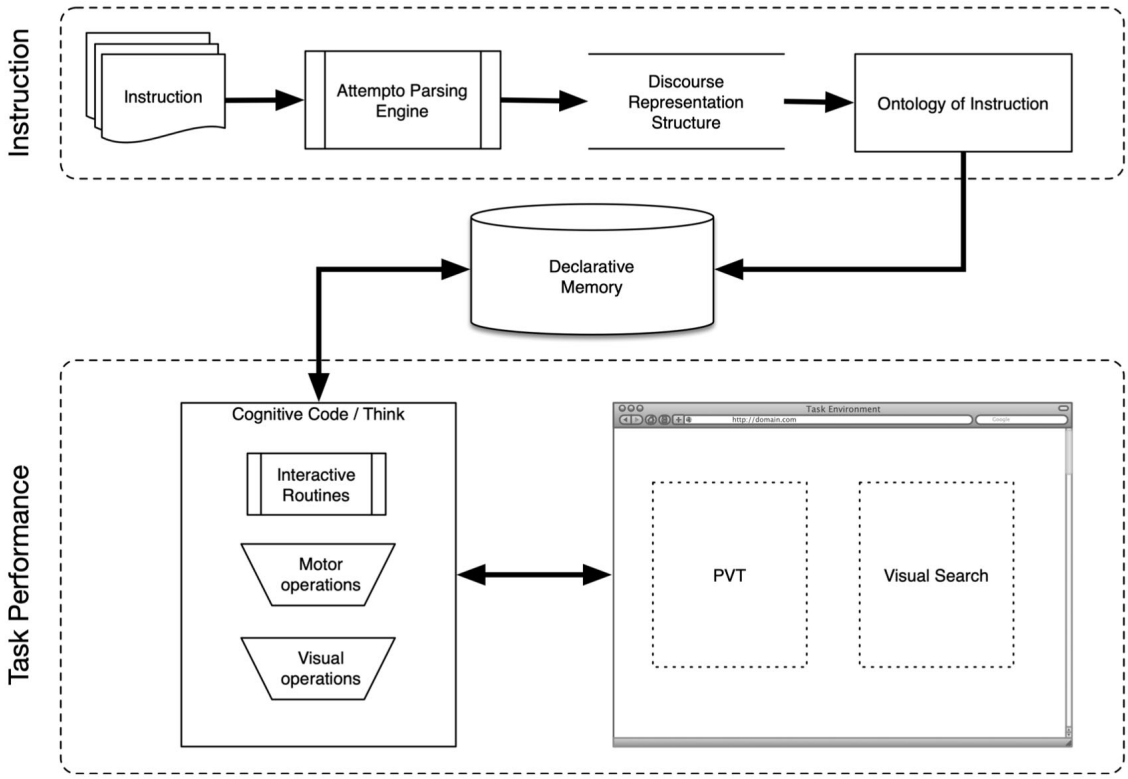


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

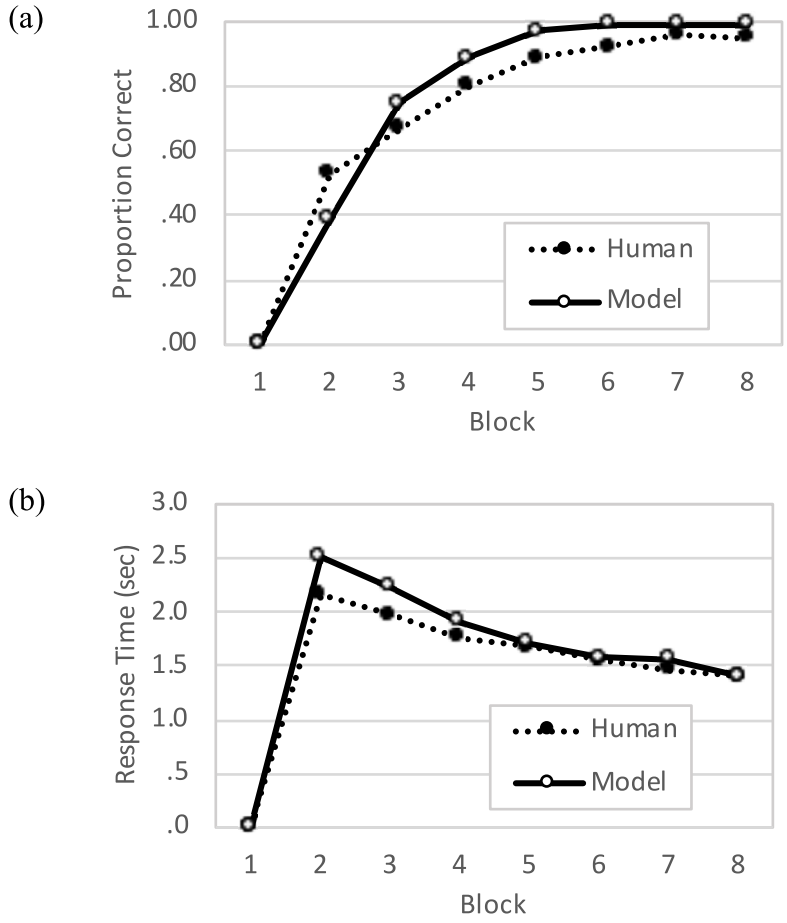
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

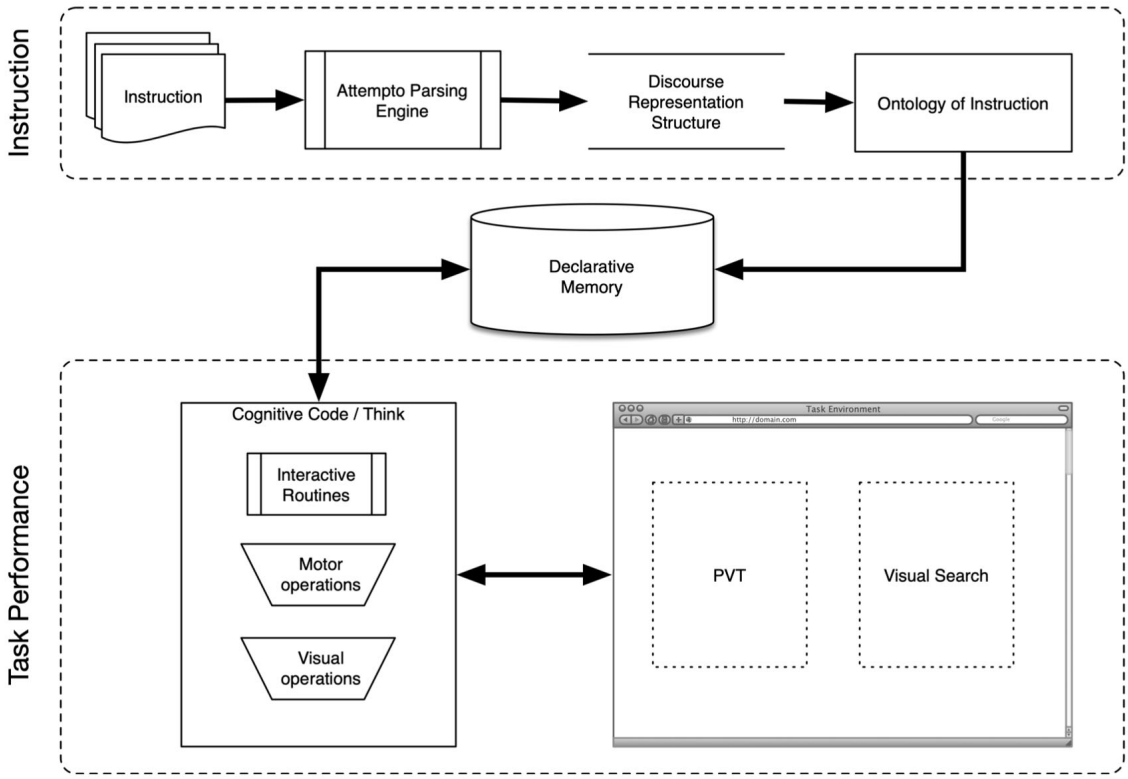


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

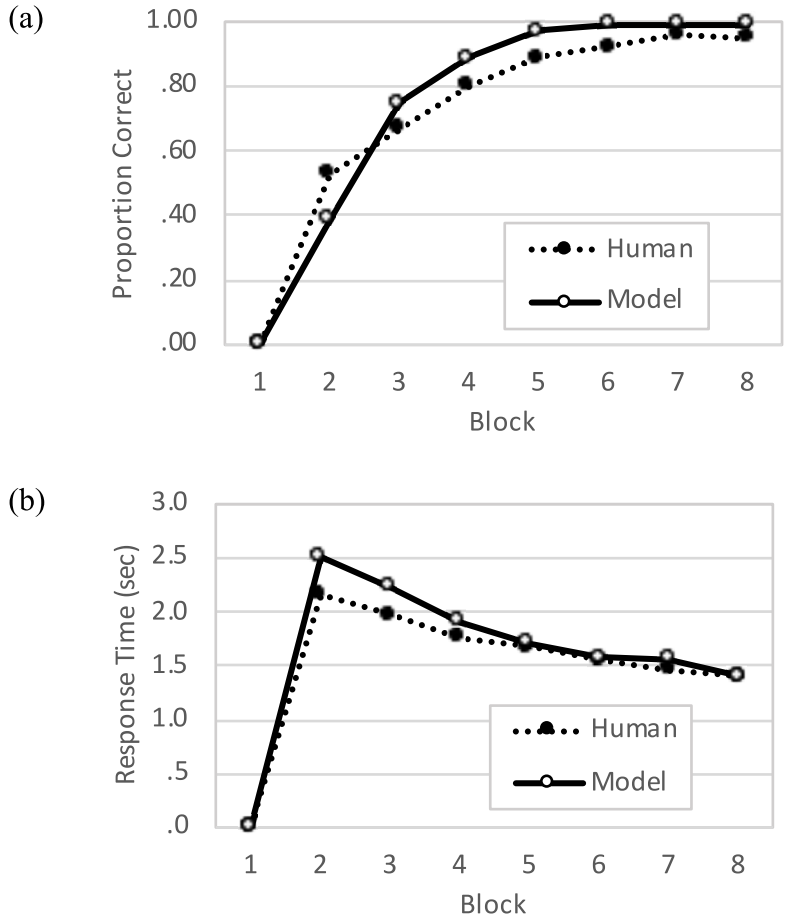
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

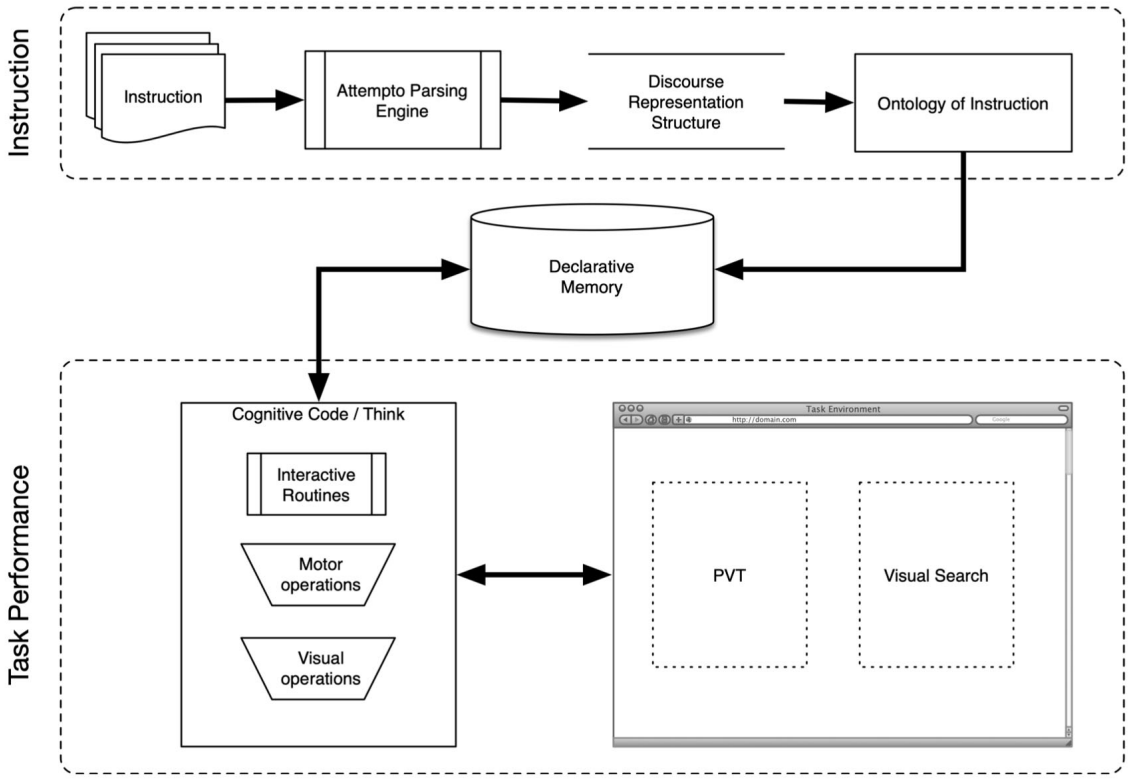


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.

FINAL REPORT

Toward Undifferentiated Cognitive Agents: Translating Instructions to Knowledge

Award: FA9550-18-1-0371

Investigator: Dr. Dario D. Salvucci
Drexel University
Email: salvucci@drexel.edu
Phone: 215-895-2674

Sponsor: Air Force Office of Scientific Research
United States Air Force

Program Manager: Dr. Laura Steckman
Program Officer, Trust and Influence
Air Force Office of Scientific Research
Email: laura.steckman.1@us.af.mil

Current Report Period: August 1, 2021 – July 31, 2022
Total Grant Period: August 1, 2018 – July 31, 2022

Abstract

In this project, we are developing a cognitive system capable of independently acquiring most required task knowledge and skill to perform a set of tasks. Our proposed approach begins with the development of a foundational and generalizable cognitive system that can be transformed into a specialized cognitive agent through written instruction, interactions with its trainers, task experience, and developer intervention (when needed). In this vein, we have developed an undifferentiated agent (uAgent) that is a set of general-purpose computational cognitive capacities enabling it to read task instructions, iteratively interact with trainers to fill gaps in task knowledge, generate the requisite task knowledge from the instructions, and complete multitasking scenarios of varying complexity.

The project represents a collaboration among several institutions, including Drexel University, Kansas State University, and the Air Force Research Laboratory. The Drexel portion of the project focuses specifically on the problem of translating instructions to knowledge. Within this effort, we have developed several key contributions: (1) a unified framework within a "cognitive code" cognitive architecture that encodes realistic instructions and translates them to executable knowledge; (2) within the framework, an approach to address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking; (3) development of an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution; (4) exploration of the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures; and (5) integration of these components with the undifferentiated agent mentioned earlier. The report highlights the details of each of these directions, and provides an understanding of how these directions are integrated into the larger effort.

1 Introduction

In this project, we are investigating novel approaches to developing an *undifferentiated cognitive agent* (uAgent): a computational agent with general-purpose computational cognitive capacities to understand and execute instructions for a variety of tasks, thus differentiating and specializing the agent's knowledge for performance of a new task. In order for a such an agent to learn from instructions, it must parse the instructions, store them for future use, and then generate the requisite procedural and declarative knowledge for performing the instructed task. Furthermore, the agent must generate a mapping between concepts within the task and use these to define/derive constructs for situation maintenance within the identified task.

Our work at Drexel University is one of a set of three tightly connected (and jointly proposed) projects, the other two being "Toward Undifferentiated Cognitive Agents for Diverse Specializations" (Air Force Research Laboratory; PIs: Myers, Stevens, & Maruyama), and "Toward Undifferentiated Cognitive Agents: Determining Gaps in Comprehension" (Kansas State University; PI: Hitzler). This report focuses on the progress made at Drexel, but in each section, we also provide information about the integration of these efforts into the larger project.

2 Accomplishments

2.1 Research Objectives

The overarching research objective for our project has remained the same throughout the project period: to research and develop novel approaches to translating increasingly complex instruction sets into knowledge an agent can use to complete a task. Within this overarching objective, we have worked toward addressing five major objectives:

- **RO1**: Develop a theoretical understanding and associated computational model of instruction following that encodes realistic instructions and translates them to executable knowledge.
- **RO2**: Address key challenges related to interpreting English-like instructions, such as anaphora resolution, grounding, and interactive instruction-taking.

- **RO3**: Develop an associated learning system that receives and learns instructions gradually over time and with practice, moving from more declarative (slower) execution to more procedural (faster) execution.
- **RO4**: Explore the implications of these enhancements with respect to teachable agents using cognitive code and cognitive architectures.
- **RO5**: Integrate, with project partners, all of the components into an undifferentiated agent that can understand and execute instructions, specializing its knowledge for differentiated cognitive tasks.

The next section provides details of the approaches and accomplishments made toward each of these research objectives.

2.2 Specific Accomplishments

RO1: Theory and Model of Instruction Following

Our theoretical approach to instruction following builds on a great deal of prior work, especially research on instruction following in the context of computational cognitive architectures such as ACT-R and Soar. In this approach, instructions are received and represented cognitively as declarative knowledge elements (or chunks)—that is, static pieces of information represented the same way as any other static knowledge element.

In this project, we have significantly extended these past efforts to use a novel cognitive-code approach along with enhancements to our method of instruction following. When the agent follows an instruction, it uses task-general procedural knowledge (typically production rules) to translate that instruction to action (for instance, initiating hand and finger movements for an instruction to type a word). In doing so, the system also generates new task-specific knowledge (e.g., a new production rule that immediately types a word for a given instruction); the task-general and task-specific rules are then both available and active in memory, although with practice, the task-specific rules are typically favored over the task-general ones. The end result is a specialized agent with task-specific rules that produce more efficient behavior than the original undifferentiated agent. Because of our use of cognitive code (Salvucci, 2016) and the *Think* cognitive architecture (https://github.com/salvucci/think), cognitive functionality is embedded directly into a modern programming language (instead of specifying its own production-system programming language). The advantage of the cognitive-code approach for our purposes is that it

greatly facilitates both development and integration with external libraries and systems, with no real loss of theoretical precision.

We have included an array of basic skills (e.g., visual encoding, mouse, and keyboard actions) that are needed for our target tasks, and the model and larger system can currently perform tasks such as the psychomotor vigilance task (PVT) and basic visual search directly from instructions. This work was published and presented (Kupitz et al., 2021), and further details can be seen later in *Technical Updates* section.

RO2: Interpretation and Translation of Instruction

One necessary and critical aspect of instruction following is the translation of instructions to knowledge, and most past efforts in the cognitive modeling community have focused more on the learning process than this translation process. As a result, most models of instruction following (including our own past work) are brittle and dependent on a very particular specification of instructions.

We have extended the state-of-the-art model of instruction following in several ways to make it more flexible and robust. Specifically, our work has focused on three areas for improving robustness: (1) grounding of instruction language to knowledge representations, (2) inference of implicit instruction knowledge, and (3) dynamic interaction to clarify or augment instruction knowledge. This work incorporates these new enhancements into our cognitive-code model of instruction following, developed in the *Think* architecture, as mentioned earlier. This work has been presented in two publications (Salvucci, 2020; Salvucci, 2021) and further details are included later in the *Technical Updates*.

RO3: Instruction Learning

Another hallmark of instruction following is closely tied to the human memory system: at first, a person receives instructions and remembers them in declarative memory, such that each instruction is recalled and executed step by step; later, with practice, these steps become gradually encoded in procedural memory, and can be executed more quickly, more reliably, and with greater ability to multitask (since declarative memory is less frequently relied upon). Although many aspects of cognitive-code models are easier than their production-system cognitive model counterparts, modeling learning is more difficult because of the rigid code base in a standard programming language.

Over the course of this project, we have developed a novel approach to modeling learning in cognitive code: in summary, the model represents both the declarative and procedural knowledge and allows them to compete with a utility function (much like the ACT-R approach). We have tested this new approach with simple psychological domains and then integrated the new approach into the larger system (see Kupitz et al., 2021).

RO4: Possible Extensions to Teachable Agents

Although this research objective was not explicitly stated in the original project proposal, we have been exploring the implications of our instruction-following work for teachable agents. The idea of teachable agents is a close relative of the undifferentiated agents focused on here, in that we would like an agent that can be taught new skills and that can then execute these skills.

Much of our work on translating instructions, learning to compile them into procedural skill, and executing them could potentially transfer directly to teachable agents. One important difference, for our purposes here, is that our project has focused primarily on psychological tasks (e.g., PVT) whereas teachable agents focus primarily on common useful tasks. We have built a prototype agent that can accept instructions and interact with a web browser for some common task—for instance, scheduling a COVID-19 vaccine on a pharmacy web site. This work was presented at the 2021 ACT-R Workshop (Salvucci & Engimann, 2021), and although it represents preliminary work, it nicely demonstrates the potential applications of our work on uAgents with that of teachable agents.

RO5: Integration with the Undifferentiated Agent

The various research components described above, particularly the first three (RO1-RO3), are important both in their right and as part of the larger integrated project. The full system integration has been an ongoing collaboration, with all of us sharing a code repository and Slack discussion groups to encourage communication (especially during remote work periods).

We have had numerous successes in our integrated work. To highlight one significant example, one of the most interesting integrated components arises in the model's knowledge and memory module: whereas our Think architecture has its own memory theory and implementation (inherited from ACT-R), in this project, we have largely replaced this memory system with the ontology-centered knowledge base primarily developed by our Kansas State partners. Our AFRL partners have also been critical in implementing the task domains and exploring how we might

search for model parameters and further optimize the model within the ARES system. The full integration is included in a recent publication (Kupitz et al., 2021).

Project Collaboration

To further all the research objectives above, we have worked closely with our project partners to ensure that the individual research components also work tightly in the context of the full undifferentiated agent. From a logistics standpoint, all project participants from all three projects have held a weekly meeting, including remote participation where required, to plan joint work and to report on ongoing project work. Selected participants furthermore had several additional (often more technical) meetings to address particular issues as they arose, especially when the implementation of the instruction-following system needed critical interfacing with the other aspects of the project (e.g., the necessary knowledge representations). We have been publishing the major lines of work in appropriate major venues, such as the *Topics in Cognitive Science* journal and the International Conference on Cognitive Modeling, and disseminating the work elsewhere in smaller venues (e.g., the ACT-R Workshop). Beyond our weekly project meetings, we have several major multi-day meetings (in-person before the COVID pandemic, then virtual) that significantly advanced the work and collaborations.

3 Impacts

Within the discipline of cognitive science, perhaps the most significant impact of our work is in the development of *cognitive code* as a framework for cognitive modeling. This project represents the first major attempt to apply cognitive code for a large-scale cognitive modeling effort (with the integrated undifferentiated agent). Our success thus far offers promise that cognitive code will be a useful way to produce psychologically plausible cognitive models with much less effort than needed by (for example) modern production-system architectures. This has impacts outside of cognitive science as well, since cognitive code explicitly aims to open up modeling to people outside the traditional modeling community—allowing them to build models in a span of minutes instead of hours or days needed by traditional architectures.

Our project has also had impact on the development of human resources, primarily in terms of training the next generation of educators and researchers. The project has provided significant funding to a Drexel Ph.D. student who has helped to develop the cognitive-code

framework and, using this, has built her own framework for a teachable agent via a common web browser. In the past year she has completed her thesis proposal (on teachable agents as related to this project), and she was a coauthor on one presentation, with additional work and publications in progress.

The impact of the project on society has been indirect thus far, but we believe that our work on undifferentiated agents and teachable agents offers great promise for the future. The ability of computers to learn from, and team with, human partners will be a critical step toward broadening the usefulness of computing in the larger society. Our work on instruction following and undifferentiated agents provides this focus, allowing for more rigorous and complete understanding of human instructions such that a computer can interactively learn from humans and perform trained tasks. As such teachable agents become more commonplace, computers will be able to help people with certain aspects of their own tasks, freeing up their human partners to focus on other aspects of the tasks in which they excel.

4 Changes

In general, our project has largely proceeded as planned. The COVID-19 pandemic forced much of our work at Drexel University to a remote setting, but most of this work (e.g., the computational modeling) could be successfully done remotely. However, one aspect of the pandemic that changed our larger project significantly was our reliance on the collection of experimental data: our project partners at AFRL had planned to collect data on humans following instructions in various forms which was to help us validate the psychological validity of our own models, and unfortunately AFRL's data collection became infeasible due to work requirements. Thus, we submitted and were approved for a 12-month no-cost extension that gave us at Drexel additional time to codify the models, publish them to a software repository, and work on final publications.

5 Technical Updates

With respect to technical progress towards the project goals, the main achievements to date are as follows:

- Major enhancements to Think, an alternative approach to coding ACT-R capabilities, as developed by our project at Drexel.
- Significant modeling work on flexible instruction following, focusing in particular on (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages.
- Application of models to two component tasks within the ISR-MATB task suite, namely the psychomotor vigilance task (PVT) and a visual-search task.
- This application utilizes ACE (Attempto Controlled English) as an initial language of instruction and converts this information into a robust knowledge base based on a rigorous ontology for the task (developed by partners at Kansas State).
- Development of additional theories and models of a teachable agent that can accept instructions from a user, and interactively clarify and execute these instructions on a standard external web site.

The next sections highlight the technical details of the above, in line with our research objectives described earlier.

5.1 Interactive Grounding and Inference in Instruction Following

[Note: The work highlighted in this section was done primarily at Drexel, with its focus on aspects of translating instructions to knowledge and execution.]

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R and Soar. More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor. This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions.

We have explored several ways in which instruction following can be made more flexible and robust (Salvucci, 2020; Salvucci, 2021). Specifically, this work examines three areas for

improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python).

Modeling with Cognitive Code

Past research efforts related to learning by instruction have traditionally developed cognitive models within the framework of production-system cognitive architectures (e.g., ACT-R: Anderson, 2007; Soar: Laird, 2012; Laird, Newell, & Rosenbloom, 1987) which use condition-action production rules to represent procedural skill. While production systems offer several benefits as a modeling paradigm—including modularity of skills and potential mapping to brain regions (see, e.g., Anderson et al., 2004)—they typically require many hours of training and practice before one can develop sufficient expertise to build useful models. Instead, the models described here embody a different approach that uses *cognitive code* (Salvucci, 2016) as a computational framework, and specifically, the Think framework (https://github.com/salvucci/think) as a sample instantiation of the idea of cognitive code. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.). Before exploring the fuller model of instruction following in the next section, we first present a basic model for a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures.

The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit

and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
  self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type word, and when it is found, encodes the visual object in the word variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2011). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory. Think's temporal predictions for these various processes are derived from those of the ACT-R cognitive architecture (Anderson, 2007).

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
  self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with production systems, whose syntax and semantics are not nearly as familiar to programmers today. Cognitive

code also takes advantage of common programming idioms—for instance, returning *None* for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'To perform a task'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like

chunks—for example, *WaitFor*(*word*) or *If*(*Recall*(*digit*, *word*), *Type*(*digit*)) for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the *WaitFor*(*word*) chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the word. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in reality these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple, straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. An earlier effort (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest

pseudo-English instructions (e.g., ‘Wait-for visual-change’). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe several ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., ‘the green rectangle’), prepositional phrases (e.g., ‘the green square to the left of the blue square’), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as ‘Wait for a word’ and builds a declarative memory chunk *WaitFor*(*word*) as a mental representation of the phrase. The concept of word is grounded to the next visual object that appears to the model, and the model will store the mapping from word to this object in the current context (analogous to ACT-R’s imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to “point out” visual information—for instance, hearing the phrase ‘Read the letter’ while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model’s simulation run, actions such as ‘Wait for’ or ‘Read’ will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to ‘Wait for a digit’ and then later to ‘Type the number’. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both digit and number to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person’s general knowledge (not something learned during the task). When a term is discovered that cannot be grounded— e.g., ‘Type the number’ when the model has not yet seen a number—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the digit, and thus it can search for and find an interpretation whereby ‘number’ and ‘digit’ refer to the same object.

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter ‘C’ among a set of distractors (such as the letter ‘O’). Pronoun resolution—here, resolving the meaning of the word ‘it’—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same ‘C’ mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object ‘C’, and later noting that it must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model’s parsing and interpretation.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions may arise in inclusion or exclusion of the steps themselves. In particular, some steps may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to 'Find,' 'Move to,' and finally 'Click on' the desired target, plus an explicit 'Repeat' step; (b) shorter instructions that skip the 'Move' step; and (c) even shorter instructions that only direct the participant to 'Click on the 'C'' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse

and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the 'Click' action requires the 'Move' action, which in turn requires the 'Find' action—and thus the single instruction 'Click on the 'C'' triggers the same sequence of actions as the equivalent three steps in Table 4(a). Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final 'Repeat' step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to Taatgen et al.'s (2008) work using preconditions and postconditions, although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

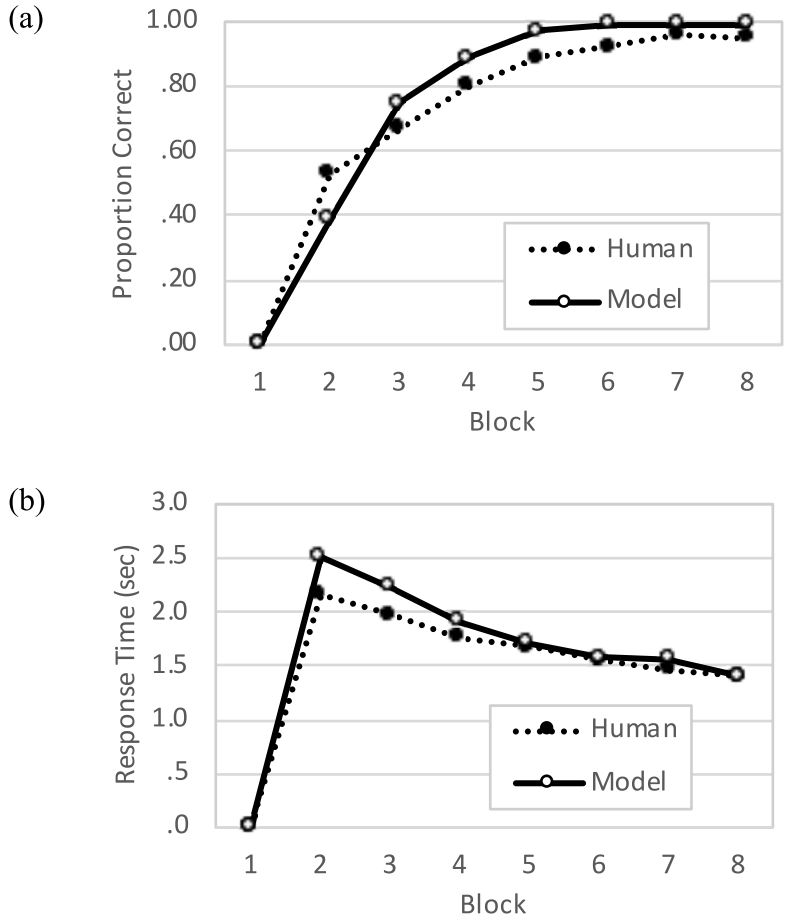
As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a single word).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a digit but then is instructed to 'Type the number.' As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling 'number' as a possible synonym of 'digit.' But what if this synonym pair is not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., 'digit' and 'target')? If no synonym is available, the model stops and asks the teacher a question such as, 'Which is the number?', and waits for a response via its audition module. When the response is given—e.g., 'the digit'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This work has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory primarily driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.



5.2 Toward Undifferentiated Cognitive Models

[Note: The work highlighted in this section represents a collaborative integration between Drexel and other partners to integrate the work in 5.1 with the theories and models of the larger project; nevertheless, we focus here on the components for which Drexel took the primary lead, notably the embedding of the declarative and procedural memory systems into the larger uAgent.]

Autonomous systems are a new frontier for socio-technical advancement. Such systems will be required to team with humans, potentially operating at the level of peers and not just

subordinates. One such autonomous synthetic teammate (AST) demonstrated that they can be included in teams without detriment to teams' or team members' performance (McNeese, Demir, Cooke, & Myers, 2017; Myers et al., 2019). Nonetheless, there remain two significant obstacles to the wider adoption of synthetic agents operating as peers or subordinates in complex environments: 1) their development and evaluation time and 2) their limited scope of transfer once developed. The AST took approximately nine years to develop plus an additional year to evaluate (Ball et al., 2010; Rodgers, Myers, Ball, & Freiman, 2013), and yet it would require further research and development to adapt it to perform a different task within the same domain and even more to adapt it to an entirely new domain.

Instructions are a ubiquitous part of the human experience. They provide guidance through a space of potential states to a solution (i.e., the problem space; Newell and Simon 1972). Without instruction, one is free to roam about the problem space freely in an attempt to find, or discover, the solution. Instruction also plays a critical role in our ability to advance as a civilization: calculus, laws of physics, and other advanced domains do not need to be re-discovered by each generation, but are taught through instruction. Although learning is sometimes characterized as the acquisition of all skills needed for a given task, in fact, the learning of complex tasks more typically reflects the integration of already-known processes (e.g., interactive routines; Gray 2008) in novel ways (Gray, Sims, Fu, & Schoelles, 2006) – of which one way is through instruction (Salvucci, 2013).

Recent advances have demonstrated the ability to turn a set of instructions into declarative knowledge that is then used to enable performance across paradigms of different complexity: ranging from those typically used by experimental psychology to dialing while driving an automobile (Salvucci, 2013). In a similar vein, Kirk, Mininger, & Laird (2016) have successfully demonstrated the ability to train robots on novel tasks through direct interaction. The objective associated with the presented research is to leverage past work on instruction learning to address the development and transfer issues, simultaneously. Specifically, we propose a generalizable, undifferentiated agent (uAgent) that can learn a new task relatively independently through written instruction and be trained to a desired level of proficiency with reduced developer intervention.

To achieve these goals, the uAgent (Kupitz et al., 2021) was developed with a modular architecture, to allow for expansion into other tasks and fields with minimal burden to other researchers. The components of the architecture are instruction parsing, an ontology of

instruction, declarative memory representation, and procedures for accomplishing the instructed tasks. Each of the uAgent components are discussed in the following sections, followed by their integration as a single system. Finally, a case study on development times relative to current approaches to model development times is presented.

Instruction Parsing

Though many advances have been made in the field of natural language processing (NLP), it still remains a challenging problem to extract complex rules and meanings out of text. To make the problem of parsing text more tractable, we use a controlled natural language - Attempto Controlled English (ACE; Fuchs, Kaljurand, & Kuhn, 2008). A controlled natural language is a language that permits only a subset of grammatical constructions available in natural language (in this case, only present perfect tense, no use of second person, and a very specific syntax for commands). These restrictions make it possible for software (e.g., the Attempto Parsing Engine, APE; Fuchs et al., 2008) to automatically parse sentences written in the controlled language into logical statements called discourse representation structures (DRS). These structures approximate first-order logic.

The requirements of the Instruction Parsing module are as follows. First, the language requirements of incoming instructions must be specified (here, we use the ACE controlled language). Second, these instructions must be processed into a form compatible with the target declarative memory system structure to be used by the acting uAgent. In the current system, we begin with plain English instructions of a task of interest. Two types of tasks we are currently working with include basic experimental psychology tasks - psychomotor vigilance (Dinges & Powell, 1985) and visual search (Treisman & Gelade, 1980) - and a material engineering task in which an individual guides a set of experiments with a 3D printer (Nikolaev et al., 2016). In both cases, instructions are re-written by hand into sentences that follow the rules of the ACE language. Then, the instructions are provided to APE 1 to translate the ACE sentences into DRS structures, which are then integrated into a declarative memory structure based on an ontology of instruction.

Table 3: PVT instructions in English and ACE.

English:
You will be seated in front of a computer screen.
A letter will appear in the middle of the screen.
When you see the letter, press the spacebar.
ACE:
p:psychomotorVigilance is a task.
There is a screen.
There is a letter.
There is a subject.
The n:spacebar is a button.
If the task is active then the subject v:watchesFor the letter and the letter v:appearsOn the screen.
If the letter v:appearsOn the screen then the subject presses the n:spacebar. The task is active.

Ontology of Instruction

The primary function of the Ontology module is to directly formalize the structure of information necessary to complete the desired tasks and goals of the uAgent. This furthers the goal of the uAgent as a whole, as it provides the foundation for the structure and relationships within the declarative memory system used by the uAgent (see Figure 1).

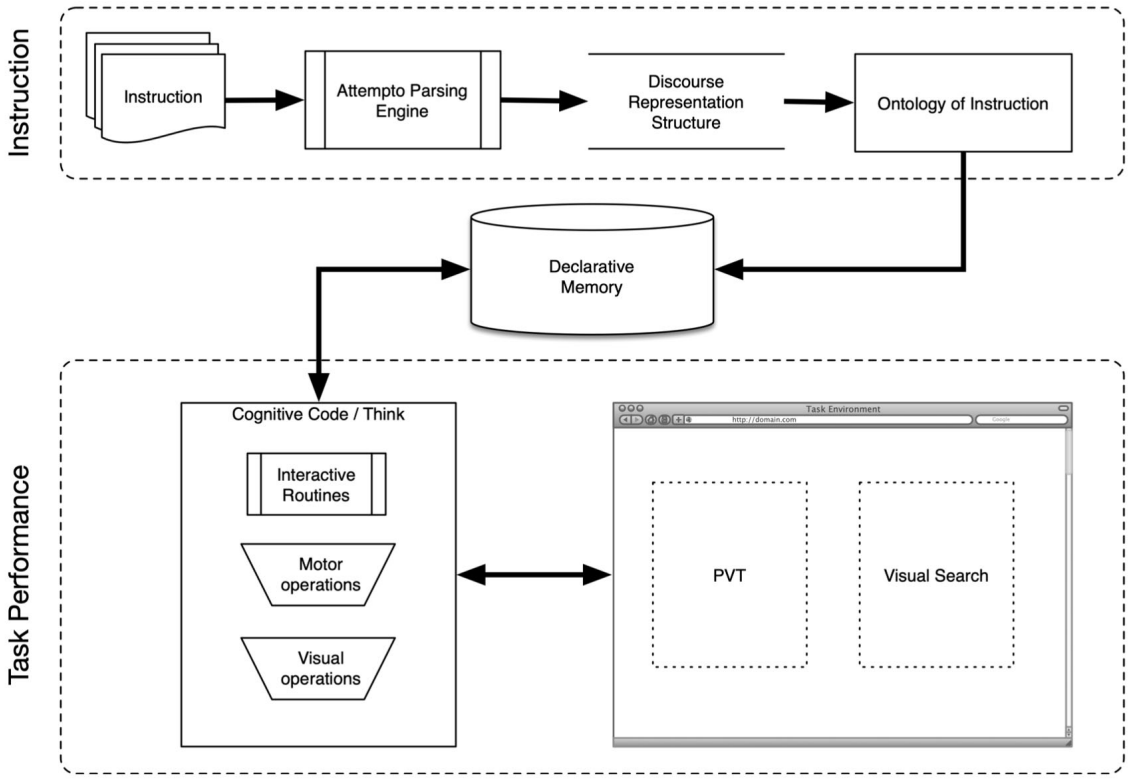


Figure 1: Architectural components of the undifferentiated cognitive agent (uAgent).

In order to create a system that is both generalizable and able to correctly handle diverse types of instructions, an ontology was created that is capable of representing instructions for a cognitive agent task. This instruction ontology can be used to directly inform relationships among tokens of knowledge within a cognitive agent performing tasks. Further, it can be leveraged to derive a semantically anchored declarative memory system for long-term storage for knowledge, such as a knowledge graph (Noy et al., 2019). It can also support experiment design, irrespective of any agent, by providing a structured basis for evaluating the content and design of similar tasks. Additionally, because an ontology contains a precise axiomatization of the knowledge it is supposed to represent, deductive reasoning techniques can be applied to detect possible gaps or errors in instructions. Further information regarding the ontology can be found in (Eberhart et al., 2020).

The ontology was developed to represent the relationships between steps, items, actions, instructions associated with tasks relying on a graphical user interface. To ensure a potentially high degree of complexity in instructions, the multistage Intelligence, Surveillance, &

Reconnaissance Multi-Attribute Task Battery (ISR-MATB) task (Frame et al., 2019) was used as an example task when developing the ontology. Because it has multiple interconnected cognitive tasks, using the ISR-MATB aids in the development of an undifferentiated representation of instruction knowledge. The ontology was produced by following the Modular Ontology Modeling (MOMo) methodology, outlined in (Krisnadhi & Hitzler, 2016; Hitzler & Krisnadhi, 2018; Shimizu, Hammar, & Hitzler, 2021), and is designed to ensure high quality and reusability of the ontology. The adaptability required to model the ISR-MATB task, together with the modular techniques used to create it, mean that the ontology can very easily be adapted for use in new tasks.

Currently, DRS items from instruction are obtained as input to the ontology whenever an agent begins learning through instruction (see Figure 1). The DRS structured information is then available to an agent during a task, and additional knowledge that the agent acquires can be added to supplement this. As new tasks are implemented and tested this process is simple to extend to encompass new types of knowledge, since the structure of the ontology and the format of input data is agnostic to the actual content of the knowledge represented.

To summarize, the ontology leverages information theory and formal logical structures to ensure that pertinent information is assimilated in the most reasonable, orderly fashion possible from a theoretical standpoint. Importantly, this ensures that any future expansions of the uAgent into additional research fields and tasks will be expedited, as any additional pertinent information can be distilled directly into the most useful form through the ontology and into the uAgent's declarative memory system.

Declarative Memory System

The declarative memory system of the uAgent contains information associated with parsed instructions, prior knowledge, and a controlled vocabulary connecting verbs to known procedures in the procedural system. The approach taken to represent the declarative memory system was a knowledge graph (Noy et al., 2019), which describes facts, actions, objects of interest, and the relationships between them. The uAgent here stores a knowledge graph built from declarative chunks. Specifically, it incorporates chunks that, using ACT-R-like slot-value pairs (Anderson, 2007), links knowledge together in a graph by having one chunk's slots include other chunks as values for those slots. As such, the representation is flexible enough to

incorporate all the declarative knowledge needed in the instructions for our purposes, including not only basic actions but also conditionals and sequences of actions.

A declarative memory system structured as a knowledge graph requires connections to real action/observable behavior to ground the information in the actions available within the instructed task. Without grounding, the agent can have all of the available information about the task but no way to observe or interact with its task environment. To this end, a controlled vocabulary (CV) was introduced to map verbs onto concepts or actions. For example, a CV entry for "search" could map onto an interactive routine (Gray, 2008) instructing the agent to attend a location, locate an item there, and encode it. Within the ACT-R paradigm, this led to creating a new class of chunks for CV entries. These chunks contain the CV term and map to a production or set of procedures built into the agent prior to instruction, thereby grounding that term onto a known set of actions (Ji, van Rij, & Taatgen, 2019).

Novel task strategies can be constructed using these grounded interactive routines, thereby allowing the agent to interact with an environment for which it was not specifically designed and perform tasks without needing to have a whole set of bespoke procedures and knowledge built into it. Altogether, the CV defines the set of verbs which are already grounded to behavior(s); in essence, it represents the agent's knowledge of general behaviors a priori. Accordingly, by relating the CV to task appropriate interactive routines, we ensure the knowledge is inherently grounded to the environment.

As a module, the knowledge graph serves as the basis of the uAgent memory: it contains information pertinent to the instructions given, but processed through the lens of overall task knowledge it should have beforehand (i.e. the Ontology). It must follow the format of the structures provided within the formal ontology, and further, should use a defined controlled vocabulary to map those terms onto active agent behaviors where appropriate.

Procedural Memory System

Given the above declarative memory structures for representing instructions, the system needs to ground concepts to simulated actions via interactive routines (i.e., embedded or learned procedural knowledge). Models developed in cognitive architectures such as ACT-R (Anderson, 2007) or Soar (Laird, 2012) typically use production systems to represent this procedural knowledge. Here, we take a different approach, using cognitive code (Salvucci, 2016) to maintain and execute procedural knowledge. Cognitive code embeds procedural knowledge into a common

programming language, facilitating the development of model code while maintaining the most important properties of human-like abilities and limitations inherent to any cognitive architecture. Specifically, we are using the Think architecture 2, which incorporates declarative and procedural concepts taken primarily from ACT-R and provides them for easy use via the Python programming language.

Several components of this project have led to important extensions of Think's code base. One extension involves the integration of traditional declarative memory with Think execution. The default Think code base includes a declarative memory module that embodies ACT-R's core theory of memory (Anderson, 2007). For this project, we bypass this traditional memory module, and instead use the ontology and knowledge graph described earlier as the model's primary long-term declarative storage. The Think procedures still maintain short-term declarative items, namely those that comprise the current "context" during execution (i.e., information that would traditionally be stored in ACT-R's imaginal buffer).

Besides this integration of a new type of declarative memory, the other critical extension of Think's code base relates to the realization of procedural learning. Although cognitive code can often be made to operate in ways very similar to traditional production systems, a critical difference is that cognitive code cannot (in most cases) be constructed during simulation as some architectures have done with procedural learning. For example, ACT-R's production compilation mechanism (Taatgen & Lee, 2003) transforms declarative instructions into procedural form which eventually leads to gradual learn of new procedures; the most critical aspect of this learning is that, at first, a model must perform a declarative retrieval to remember the learned instruction before executing it, but later, the compiled instruction (in the form of a production rule) skips the retrieval and simply executes the associated action. Although Think does not create new code on the fly in the same way, we have augmented its capabilities by adding procedural learning that captures the essence of ACT-R's production compilation—specifically, in performing declarative retrievals early in learning (which take additional time and may fail), and then skipping these retrievals later in learning (leading to gradual speedup and eventually fast performance).

As a module, the cognitive code contained with Think serves as the "actual" uAgent, so to speak – it represents the system which is deciding and acting upon the best course of behavior during any task. In theory, this could be replaced with any number of cognitive architectures, provided they are capable of using the prespecified knowledge graph structures to serve as the

basis of memory, and further, have a correctly specified controlled vocabulary to map that knowledge graph onto the behaviors known to the system a-priori.

System Integration

To develop the uAgent with an adaptable framework going forward, we used a modular design approach (Bryson, 2000). In particular, this capitalizes on the interdisciplinary nature of the researchers involved while simultaneously minimizing the overall burden of coordination. To that end, during development the fundamental uAgent capabilities were segregated into discrete modules. Overall integration of these modules was then assigned to a few individuals, with the entire research team meeting to discuss overall design strategies as appropriate. Of note, this approach also allowed for a degree of asynchronous development across the research teams involved, thereby reducing the project coordination burden significantly. In addition, the modular approach ensures that the uAgent will be adaptable to other fields of research and task performance, as future research can adapt the uAgent by focusing on a specific uAgent module where appropriate. We now move on to discuss the primary modules of interest in the uAgent.

Given the interdisciplinary nature of this research, we first settled on the use of the open source Python as the primary programming language, integrating each individual uAgent module into one coherent system. In particular, this allows us to utilize the Think system (Salvucci, 2021) in order to simulate both the uAgent behavior, and the environment in which is it actively behaving. Further, whenever these separate modules are expected to interact directly, we worked to determine the best overall form of interface and information exchange to facilitate ease of integration and future expansion. To that end, we now note the primary interface decisions we made during said development.

First, we concluded that the ontology of instruction should serve as a form of blueprint for the knowledge graph. This ensures that the Instruction Parsing module will produce structures that can be assigned to knowledge graph structures where appropriate. Effectively we are leveraging the relations inherent to the ontology in order to improve the capabilities of the instruction interpretation; in essence, the uAgent can make informed assumptions about the informational structure while processing any incoming instructions.

Similarly, to ensure the agent is capable of acting on those instructions, we concluded that the knowledge graph module should also consider a controlled vocabulary representing the behaviors found within the Think cognitive agent. This controlled vocabulary is essentially the

actions that the think uAgent is capable of performing in the current environment. In essence, we ensure that the knowledge graph structures which serve as the basis of the Think agent memory also have a direct mapping onto Think behaviors where appropriate.

Altogether, we integrate each of the uAgent modules into a coherent end-to-end system, and explicitly define the interface requirements necessary to ensure the system can take instructions as input and produce human behavior with high fidelity.

Case Study

As a proof-of-concept for the approach, we built an end-to-end system that takes ACE instructions of cognitive tasks commonly used in basic research – psychomotor vigilance and visual search – converts the instructions into a knowledge representation capable of performing the task, and then performs the task in a simulated environment.

As an exercise to determine if the current approach could save time with respect to building a traditional ACT-R model, we compared the amount of time it took to build a model of a set of cognitive tasks with the amount of time it took to write ACE instructions of the same task. The task we used was a novel task battery that includes a set of commonly used experimental psychology tasks (Frame et al., 2019; Eberhart et al., 2020). This battery includes four subtasks - psychomotor vigilance, visual search, auditory search, and multi-cue decision-making. We a built model of the task in a Java implementation of ACT-R 6 and wrote a set of ACE instructions for it.

It took approximately 120 hours to build the ACT-R model, but only approximately 30 hours to write the ACE instructions. This exercise suggests that the present method could potentially save a substantial amount of time in developing new models and agents. Moreover, writing the ACE instructions required only a brief reading of publicly available tutorials on the ACE language, and not training and experience in writing ACT-R models, the latter of which can be substantial. In our proof-of-concept system, we showed that the ACE instructions of the PVT and Visual Search subtasks could be successfully integrated into the ontology and the agent could use this resulting knowledge to perform the task. We are working toward end-to-end demonstrations of the other two subtasks.

Conclusions & Future Work

Progress toward a modeling framework capable of being taught new tasks through written instruction was presented. As evidenced in the uAgent case study, such an approach will likely significantly reduce model and agent development times. Further, the module-based

approached toward uAgent development will facilitate the integration of other cognitive architectures by using the uAgent declarative memory as its knowledge repository.

While the uAgent shows promise as a means for teaching models how to perform new tasks, multiple challenges remain. For example, it is unreasonable to assume that the union of instruction and prior knowledge is sufficient for completing an instructed task. As a result, we have begun developing approaches for detecting and resolving gaps in a uAgent's knowledge base. This work will require multidisciplinary approaches to model development coupled with empirical investigations into when and how humans detect and resolve knowledge gaps.

In order to better understand how humans form representations from instructions and identify and resolve gaps in understanding from those instructions, we plan to conduct a human-subjects experiment using the task battery described above. We plan to teach participants to perform the tasks in the battery using either a complete set of instructions, or a set with ambiguities with respect to certain types of knowledge. We plan to use think-aloud protocols to track how participants extract knowledge from these instructions and how they detect and resolve uncertainty. We believe this will provide insights in how to improve the undifferentiated model's knowledge acquisition.

6 References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*, 326.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036.

Ball, J., Myers, C., Heiberg, A., Cooke, N. J., Matessa, M., Freiman, M., & Rodgers, S. (2010). The synthetic teammate project. Computational and Mathematical Organization Theory, 16(3), 271–299. doi: 10.1007/s10588- 010-9065-3

Bryson, J. (2000). Cross-paradigm analysis of autonomous agent architecture. Journal of Experimental & Theoretical Artificial Intelligence, 12(2), 165–189.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 2–9).

Dinges, D. F., & Powell, J. W. (1985). Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers, 17(6), 652–655.

Eberhart, A., Shimizu, C., Stevens, C., Hitzler, P., Myers, C. W., & Maruyama, B. (2020). A Domain Ontology for Task Instructions. In B. Villazo´n-Terrazas, F. OrtizRodr´ıguezm, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge graphs and semantic web. second iberoamerican conference and first indo-american conference, kgswc 2020 (pp. 1–13). Me´rida, Mexico: Communications in Computer and Information Science, vol. 1232.

Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., & Boydstun, A. (2019). Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In Presented to the annual meeting of the psychonomic society conference, montreal, qc, canada, november 2019.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) (Vol. 5224 LNCS, pp. 104–124). doi: 10.1007/9783-540-85658-03

Gray, W. D. (2008). The Interactive Routine as Key Construct in Theories of Interactive Behavior. In V. Sloutsky, B. Love, & K. McRae (Eds.), 30th annual meeting of the cognitive science society (p. 127). Austin, TX.

Gray, W. D., Sims, C. R., Fu, W.-T., & Schoelles, M. J. (2006). The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. Psychological Review, 113(3), 461–482.

Hitzler, P., & Krisnadhi, A. (2018). A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR, abs/1808.08433. Retrieved from http://arxiv.org/abs/1808.08433

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*, 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of Artificial Intelligence Research*, *3*, 271–324.

Ji, M. Y., van Rij, J., & Taatgen, N. A. (2019). Discoveries of the algebraic mind: A PRIMS model. Proceedings of ICCM 2019 - 17th International Conference on Cognitive Modeling, 71–76.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Krisnadhi, A., & Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, & V. Presutti (Eds.), Ontology engineering with ontology design patterns – foundations and applications (Vol. 25, pp. 3–21). IOS Press.

Kupitz, C., Eberhart, A., Schmidt, D., Stevens, C. A., Shimizu, C., Hitzler, P., Salvucci, D. D., Maruyama, B., & Myers, C. (2021). Toward undifferentiated cognitive models. In *Proceedings of the 19th International Conference on Cognitive Modeling*.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2017). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872081774322. doi: 10.1177/0018720817743223

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., et al. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*, 3–14.

Newell, A., & Simon. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice Hall.

Nikolaev, P., Hooper, D., Webber, F., Rao, R., Decker, K., Krein, M., . . . Maruyama, B. (2016). Autonomy in materials research: a case study in carbon nanotube growth. Nature Partner Journal: Computational Materials, 2(1), 16031. Retrieved from http://www.nature.com/articles/npjcompumats201631 doi: 10.1038/npjcompumats.2016.31

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., & Taylor, J. (2019). Industry-scale knowledge graphs: Lessons and challenges. Communications of the ACM, 62(8), 36–43. doi: 10.1145/3331166

Rodgers, S., Myers, C., Ball, J., & Freiman, M. (2013). Toward a situation model in a cognitive architecture. Computational and Mathematical Organization Theory, 19, 313– 345.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*, 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

Salvucci, D. D. (2020). Interactive grounding and inference in instruction following. In Proceedings of the 18th International Conference on Cognitive Modeling (pp. 236-240).

Salvucci, D. D. (2021). Interactive Grounding and Inference in Instruction Following. To appear in *Topics in Cognitive Science*.

Salvucci, D. D., & Engimann, J. (2021). Explorations of ACT-R, cognitive code, and teachable agents. Presented at the 2021 ACT-R Workshop.

Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web. (Under Review.)

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*, 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*, 61–76.

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12, 97–136.