AFRL-RI-RS-TR-2022-146

LIFELONG LEARNING OF PERCEPTION AND ACTION IN AUTONOMOUS SYSTEMS

TRUSTEES OF THE UNIVERSITY OF PENNSYLVANIA

NOVEMBER 2022

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-146 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
LISA M. LOOMIS
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
NOVEMBER 2022	FINAL TECHNICAL REPORT	MAY 2018	MAY 2022

4. TITLE AND SUBTITLE

LIFELONG LEARNING OF PERCEPTION AND ACTION IN AUTONOMOUS SYSTEMS

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	FA8750-18-2-0117	61101E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2GV

6. AUTHOR(S)

Eric Eaton, Satinder Singh Baveja, Kristen Grauman, George Konidaris, Erik Learned-Miller, Michael Littman, Maja Mataric, Fei Sha, Peter Stone

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Trustees of the University of Pennsylvania
3451 Walnut Street
Philadelphia PA 19104-6309

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/ RI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-RI-RS-TR-2022-146

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Under the DARPA Lifelong Learning Machines (L2M) program, we explored a comprehensive approach to lifelong learning for autonomous systems, addressing fundamental issues of continual learning and transfer across diverse tasks, scalable knowledge maintenance, self-directed learning, and adaptation to changing environments for embodied agents. The key aspects of our L2M approach include: continual learning for perception and action, transfer between diverse tasks, scalable lifelong knowledge maintenance, self-directed learning for autonomous discovery, and modeling the non-stationary distribution of tasks. We explored each of these aspects separately, developing various lifelong learning algorithms for classification and reinforcement learning settings. These developed algorithms were then integrated together via modular framework, yielding an L2M system that supports both classification and reinforcement learning tasks. We evaluated the lifelong learning performance of this L2M system using the Johns Hopkins Applied Physics Lab's MiniGrid lifelong learning benchmark, and applied this L2M system to integrated perception and action through a robotic scavenger hunt using Matterport 3D, showcasing our L2M system's ability to rapidly learn diverse tasks in unstructured environments and quickly adapt to changes.

15. SUBJECT TERMS

Lifelong machine learning, autonomous systems, reinforcement learning, service robots

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	234

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
LISA M. LOOMIS	N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 SUMMARY

Under the DARPA Lifelong Learning Machines (L2M) program, we explored a comprehensive approach to lifelong learning for autonomous systems, addressing fundamental issues of continual learning and transfer across diverse tasks, scalable knowledge maintenance, self-directed learning, and adaptation to changing environments for embodied agents. The key aspects of our L2M approach include: continual learning for perception and action, transfer between diverse tasks, scalable lifelong knowledge maintenance, self-directed learning for autonomous discovery, and modeling the non-stationary distribution of tasks. We explored each of these aspects separately, developing various lifelong learning algorithms for classification and reinforcement learning settings. These developed algorithms were then integrated together via modular framework, yielding an L2M system that supports both classification and reinforcement learning tasks.

We evaluated the lifelong learning performance of this L2M system using the Johns Hopkins Applied Physics Lab's MiniGrid lifelong learning benchmark. In comparison to single-task experts, for the benchmark's Condensed and Dispersed scenarios respectively, our results showed our system's ability to significantly speed up learning with an average forward transfer ratio of 4.18 and 3.55, and an average sample efficiency of 1.32 and 1.15. In addition to efficiency, our system also shows more effective performance than single-task experts, with positive relative performance ratios of 1.04 and 1.03, and positive backward transfer ratios of 1.12 and 1.04.

We also applied this L2M system to integrated perception and action through a robotic scavenger hunt using Matterport 3D, showcasing our L2M system's ability to rapidly learn diverse tasks in unstructured environments and quickly adapt to changes. Our results showed that our system learns classification tasks approximately twice as quickly and twice as accurately as compared to single-task experts, showing mean relative performance of 2.21 and mean sample efficiency of 1.71, while completely mitigating catastrophic forgetting. For reinforcement learning settings in this domain, our system obtains a mean performance maintenance ratio of 4.37 and mean backward transfer ratio of 1.11, which shows that our lifelong learning agent is able to learn new tasks while mitigating catastrophic forgetting.Our system also shows strong potential in leveraging past knowledge to jump-start learning for reinforcement learning tasks, with a mean forward transfer ratio of 3.11. However, tuned single-task experts were able to outperform our system on individual reinforcement learning tasks, with our system reaching a mean relative performance ratio of only 0.88. We also conducted various ablative experiments and assessment of individual lifelong learning components.

Collectively, our project produced over 110 scientific publications showcasing our work and results that fundamentally advance our understanding of and ability to perform lifelong machine learning. As two examples of this, our project developed state-of-the-art visual navigation using occupancy anticipation that won the 2020 Habitat PointNav Challenge, and a live demo showcasing real-time lifelong learning on a service robot in front of an audience during the ICRA 2022 conference.

2 INTRODUCTION

2.1 Purpose of this Report

This report documents our project under the DARPA Lifelong Learning Machines (L2M) program, covering our work in both phases 1 and 2 of the program.

2.2 Overview of Our Approach

Our project explored a comprehensive approach to lifelong learning for autonomous systems, addressing fundamental issues of continual learning and transfer across diverse tasks, scalable knowledge maintenance, self-directed learning, and adaptation to changing environments for embodied agents. The key aspects of our L2M approach include: continual learning for perception and action, transfer between diverse tasks, scalable lifelong knowledge maintenance, self-directed learning for autonomous discovery, and modeling the non-stationary distribution of tasks.

In phase 1, we explored each of these aspects separately, developing various lifelong learning algorithms for classification and reinforcement learning settings. These developed algorithms were evaluated in individual experiments.

In phase 2, we developed an integrated and modular framework for combining these above aspects into an L2M system that supports both classification and reinforcement learning tasks. The most promising of the phase 1 algorithms for each aspect were chosen to be implemented as modules within this system. We then applied this L2M system (1) to the Johns Hopkins Applied Physics Lab's (APL) MiniGrid lifelong learning benchmark and (2) to integrated perception and action through a robotic scavenger hunt using Matterport 3D, showcasing our L2M system's ability to rapidly learn diverse tasks in unstructured environments and quickly adapt to changes. During phase 2, we also continued development of the individual algorithms from phase 1, and explored other individual approaches to lifelong learning. As any of these fundamental algorithms showed promise toward our L2M evaluation, we then transitioned them into modules within the L2M system.

2.3 How This Report is Organized

Since this project encompasses a large number of algorithms (detailed in over 110 publications, see Appendix A), this report will first focus on the integrated L2M system in Section 3.1, showing the high-level view for how we constructed our lifelong learning system and how it supports the individual lifelong learning algorithms. We then describe the major individual lifelong learning algorithms developed across both phases 1 and 2 of our effort in Sections 3.3–3.4. Our presentation of results is split similarly, first exploring the evaluation of the integrated L2M system in the applications to APL MiniGrid (Section 4.2) and our Scavenger Hunt for embodied agents (Section 4.3), and then exploring the evaluation of the individual algorithms developed under this project (Sections 4.4–4.5).

2.4 Major Contributions

Our work has produced a number of major contributions, described in this section. For convenience, we group those contributions according to those methods that have been incorporated into the

integrated L2M System and other methods that have not. As detailed in Section 3.1, we chose algorithms for integration as components into the L2M system based on their maturity at the end of phase 1 and during the earlier part of phase 2, and their necessity for our applications.

2.4.1 Integrated System and Components.

An Integrated Lifelong Learning Framework We developed a modular lifelong learning system that supports both classification and reinforcement learning (RL) tasks in realistic service robot settings. The core of the system integrates factorized lifelong learning methods with the perception-action loop of a mobile robot, which we divide into separate classification and RL pipelines. The system includes additional optional modules which can be combined with the core classification and RL pipelines, including support for meta-learning, intrinsic motivation, exploration, active visual mappping, and curriculum learning. These components can be enabled or disabled depending on the problem domain, and we discuss example configurations developed and evaluated for a visual scavenger hunt application.

Deconvolutional factorized CNN (DF-CNN) for lifelong deep learning Existing work in non-deep multi-task and lifelong learning has shown success with using factorized representations of the model parameter space for transfer, permitting more flexible construction of task models. Inspired by this idea, we introduce a novel architecture for sharing latent factorized representations in convolutional neural networks (CNNs). The proposed approach, called a deconvolutional factorized CNN [4, 5], uses a combination of deconvolutional factorization and tensor contraction to perform flexible transfer between tasks. Experiments on two computer vision data sets show that the DF-CNN achieves superior performance in challenging lifelong learning settings, resists catastrophic forgetting, and exhibits reverse transfer to improve previously learned tasks from subsequent experience without retraining. DF-CNN achieved 19.2% and 7.9% improvement over a single-task learner on CIFAR-100 and Office-Home tasks, respectively, beating other multi-task and lifelong learning baselines.

Lifelong policy gradients: faster training without forgetting (LPG-FTW) Policy gradient (PG) methods have shown success in learning control policies for high-dimensional dynamical systems. Their biggest downside is the amount of exploration they require before yielding high-performing policies. In a lifelong learning setting, in which an agent is faced with multiple consecutive tasks over its lifetime, reusing information from previously seen tasks can substantially accelerate the learning of new tasks. We provide a novel method for lifelong policy gradient learning that trains lifelong function approximators directly via policy gradients, allowing the agent to benefit from accumulated knowledge throughout the entire training process. We show empirically that our algorithm learns faster and converges to better policies than single-task and lifelong learning baselines, and completely avoids catastrophic forgetting on a variety of challenging domains. On Meta-World tasks, LPG-FTW achieved a 17.5% improvement over an agent trained individually on each task, and a 533% improvement over the closest competing lifelong learning method.

Meta-Optimizer for Fast Adaptation (KFO) We develop an algorithm, META-KFO, which is able to transform the gradients of a smaller model without increasing its modeling capacity, but still resulting in better meta-learnability. We discuss and analyze our proposed META-KFO algorithm alongside a brief account of various approaches for learning to optimize. We surmise why sufficiently

large deep models can meta-learn: the upper layers have the equivalent effect of transforming the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed only of the bottom layers.

Occupancy Anticipation for Efficient Exploration and Navigation (OCCANT) State-of-the-art navigation methods leverage a spatial memory to generalize to new environments, but their occupancy maps are limited to capturing the geometric structures directly observed by the agent. We developed *occupancy anticipation*, where the agent uses its egocentric RGB-D observations to infer the occupancy state beyond the visible regions. In doing so, the agent builds its spatial awareness more rapidly, which facilitates efficient exploration and navigation in 3D environments. By exploiting context in both the egocentric views and top-down maps our model successfully anticipates a broader map of the environment, with performance significantly better than strong baselines. Our main contributions are: (1) a novel occupancy anticipation framework that leverages semantic and geometric context from egocentric RGB(D) views; (2) a new exploration policy approach that incorporates occupancy anticipation to obtain more complete maps with less exploration; and (3) successful navigation results that improve over the state of the art in apples-to-apples comparisons, including when generalizing to environments in a disjoint dataset. Our approach is the winning entry in the 2020 Habitat PointNav Challenge.

Learning Intrinsic Rewards for Policy Gradient Methods (LIRPG) The Optimal Reward Problem [6] aims to learn the parameters of the intrinsic reward such that the resulting rewards achieve a learning dynamic for an RL agent that maximises the lifetime (extrinsic) return on tasks drawn from some distribution. We propose a meta-gradient approach [7, 8] to solve the optimal reward problem. At a high-level, we sample a new task and a new random policy parameter at each lifetime iteration, and simulate an agent's lifetime using an intrinsic reward function with policy gradient. Concurrently, we compute the meta-gradient by taking into account the effect of the intrinsic rewards on the policy parameters to update the intrinsic reward function with a lifetime value function. Through analysis of the intrinsic rewards, we show how our approach can encourage exploration under uncertainty, exploit causal relationships between objects, and account for non-stationary rewards.

Learning Curriculum Policies for Reinforcement Learning (CMDP) Curriculum learning in reinforcement learning is a training methodology that seeks to speed up learning of a difficult target task, by first training on a series of simpler tasks and transferring the knowledge acquired to the target task. Automatically choosing a sequence of such tasks (i.e., a *curriculum*) is an open problem that has been the subject of much recent work in this area. In this project, we build upon a recent method for curriculum design, which formulates the curriculum sequencing problem as a Markov Decision Process (MDP). We extend this model to handle multiple transfer learning algorithms, and show for the first time that a *curriculum policy* over this MDP can be learned from experience [9]. We explore various representations that make this possible, and evaluate our approach by learning curriculum policies for multiple agents in two different domains. The results show that our method produces curricula that can train agents to perform on a target task as fast or faster than existing methods. In addition, our recent progress shows that such a learned curriculum policy for one set of tasks can be generalized to an unseen novel set of tasks [10].

Object relationships & distribution models (OBJMAP) To help situated agents solve visual search tasks more efficiently, we propose modeling object-object spatial relationships across multiple

environments. While exploring an environment, an agent should ideally leverage knowledge about objects that have already been seen to help find target objects more quickly. We address this problem by learning co-occurrence statistics between different categories of objects, building a map of the environment containing the location of all seen objects, and then combining this information to compute the probability for a target object to be present at each location on the map.

2.4.2 Other Methods.

Compositional lifelong classification (COMPCLF) and reinforcement learning (COMPRL) A hallmark of human intelligence is the ability to construct self-contained chunks of knowledge and adequately reuse them in novel combinations for solving different yet structurally related problems. Learning such compositional structures has been a significant challenge for artificial systems, due to the combinatorial nature of the underlying search problem. To date, research into compositional learning has largely proceeded separately from work on lifelong or continual learning. We integrate these two lines of work to present a general-purpose framework for lifelong learning of compositional structures that can be used for solving a stream of related tasks. Our framework separates the learning process into two broad stages: learning how to best combine existing components in order to assimilate a novel problem, and learning how to adapt the set of existing components to accommodate the new problem. This separation explicitly handles the trade-off between the stability required to remember how to solve earlier tasks and the flexibility required to solve new tasks, as we show empirically in an extensive evaluation in classification settings. We then explore a particular form of composition for RL based on neural modules and present a set of RL problems that intuitively admit compositional solutions. Empirically, we demonstrate that neural composition indeed captures the underlying structure of this space of problems. We further propose a compositional lifelong RL method that leverages accumulated neural components to accelerate the learning of future tasks while retaining performance on previous tasks via off-line RL over replayed experiences. Using composable representations in continual learning provides a performance gain of 82.5% relative accuracy when tasks are massively diverse over non-modular methods.

Unsupervised Hard Example Mining from Videos for Improved Object Detection (DETFLICK) Important gains have recently been obtained in object detection by using training objectives that focus on *hard negative* examples, i.e., negative examples that are currently rated as positive or ambiguous by the detector. These examples can strongly influence parameters when the network is trained to correct them. Unfortunately, they are often sparse in the training data, and are expensive to obtain. In this work, we show how large numbers of hard negatives can be obtained *automatically* by analyzing the output of a trained detector on video sequences. In particular, detections that are *isolated in time*, i.e., that have no associated preceding or following detections, are likely to be hard negatives. We describe simple procedures [11] for mining large numbers of such hard negatives (and also hard *positives*) from unlabeled video data. Our experiments show that retraining detectors on these automatically obtained examples often significantly improves performance. We present experiments on multiple architectures and multiple data sets, including face detection, pedestrian detection and other object categories.

Automatic adaptation of object detectors to new domains using self-training (STSL) This work addresses the unsupervised adaptation of an existing object detector to a new target domain.

We assume that a large number of unlabeled videos from this domain are readily available. We automatically obtain labels on the target data by using high-confidence detections from the existing detector, augmented with hard (misclassified) examples acquired by exploiting temporal cues using a tracker. These automatically-obtained labels are then used for re-training the original model. A modified knowledge distillation loss is proposed, and we investigate several ways of assigning soft-labels to the training examples from the target domain. Our approach [12] is empirically evaluated on challenging face and pedestrian detection tasks: a face detector trained on WIDER-Face, which consists of high-quality images crawled from the web, is adapted to a large-scale surveillance data set; a pedestrian detector trained on clear, daytime images from the BDD-100K driving data set is adapted to all other scenarios such as rainy, foggy, night-time. Our results demonstrate the usefulness of incorporating hard examples obtained from tracking, the advantage of using soft-labels via distillation loss versus hard-labels, and show promising performance as a simple method for unsupervised domain adaptation of object detectors, with minimal dependence on hyper-parameters.

Half&Half: New Tasks and Benchmarks for Studying Visual Common Sense (HNH) The general recognition of objects, people, actions and scene types has been a core focus of computer vision research. However, now that we have achieved a degree of success in these problems, it is time to define new problems that will spur us to reach the next level of visual intelligence. The development of *visual common sense* is critical to the development of intelligent agents that can be useful in dynamic, novel environments.

But what exactly is visual common sense? We suggest that the ability to make intelligent assessments of where things might be, when not directly visible, is a critical and ubiquitous capability shared by humans and other intelligent beings, and is a fundamental component of visual common sense. Humans regularly demonstrate the ability to make decisions in the absence of explicit visual cue. This sort of "intelligent search" is a prominent example of visual common sense, and we believe it represents a skill that will be essential in developing intelligent agents. Closely related to our work are earlier efforts on incorporating contextual information for visual prediction [13, 14, 15, 16]. We believe a formal benchmark on such capabilities in the most basic forms can be a valuable addition.

In this work, we formalize the problem of inferring the presence of what we cannot already see in an image. To do this, we rely on the fact that different views of an image depict the same scene. Hence, individual sections can be used as contextual cues for the other section. For this reason, we call these tasks the Half&Half tasks [17].

Efficient Lifelong Inverse Reinforcement Learning (ELIRL) Methods for learning from demonstration (LfD) have shown success in acquiring behavior policies by imitating a user. However, even for a single task, LfD may require numerous demonstrations. For versatile agents that must learn many tasks via demonstration, this process would substantially burden the user if each task were learned in isolation. To address this challenge, we introduce the novel problem of *lifelong learning from demonstration*, which allows the agent to continually build upon knowledge learned from previously demonstrated tasks to accelerate the learning of new tasks, reducing the amount of demonstrations required. As one solution to this problem, we propose the first lifelong learning approach to inverse reinforcement learning, which learns consecutive tasks via demonstration, continually transferring knowledge between tasks to improve performance. Sharing information across demonstrated tasks resulted in an reduction of approximately 65% in the recovered reward function.

Task-agnostic lifelong learning using high-level shared sets of features (SHELS) Deep neural networks (DNNs) typically fail to generalize to unseen categories in dynamic open-world environments, in which the number of concepts is unbounded. In contrast, human and animal learners have the ability to incrementally update their knowledge by recognizing and adapting to novel observations. In particular, humans characterize concepts via exclusive (unique) *sets* of essential features, which are used for both recognizing known classes and identifying novelty. Inspired by natural learners, we develop a Sparse High-level-Exclusive, Low-level-Shared feature representation (SHELS) that simultaneously encourages learning exclusive sets of high-level features and essential, shared low-level features. The exclusivity of the high-level features enables the DNN to automatically detect out-of-distribution (OOD) data, while the efficient use of capacity via sparse low-level features permits accommodating new knowledge. The resulting approach uses OOD detection to perform class-incremental lifelong learning without known class boundaries. We show that using SHELS for novelty detection results in statistically significant improvements over state-of-the-art OOD detection approaches over a variety of benchmark datasets. Further, we demonstrate that the SHELS model mitigates catastrophic forgetting in a class-incremental learning setting, enabling a combined novelty detection and accommodation framework that supports learning in open-world settings.

Benchmarking compositional reinforcement learning (CompoSuite) We created CompoSuite, an open-source simulated robotic manipulation benchmark for compositional multi-task and continual RL. Each CompoSuite task requires a particular *robot* arm to manipulate one individual *object* to achieve a task *objective* while avoiding an *obstacle*. This compositional definition of the tasks endows CompoSuite with two remarkable properties. First, varying the robot/object/objective/obstacle elements leads to hundreds of RL tasks, each of which requires a meaningfully different behavior. Second, RL approaches can be evaluated specifically for their ability to learn the compositional structure of the tasks. This latter capability to functionally decompose problems would enable intelligent agents to identify and exploit commonalities between learning tasks to handle large varieties of highly diverse problems. We benchmark existing single-task, multi-task, and compositional learning algorithms on various training settings, and assess their capability to compositionally generalize to unseen tasks. Our evaluation exposes the shortcomings of existing RL approaches with respect to compositionality and opens new avenues for investigation. On average, the single-task and multi-task agents are able to solve approximately 40% of the tasks while an agent with explicit compositional structure improves upon this and solves 92% of the tasks on the full benchmark.

Multi-actor-attention-critic (MAAC) for multi-agent reinforcement learning Lifelong learning agents may be required to cooperate and/or compete with other learning agents over the course of their lifetime. Traditional reinforcement learning algorithms are unable to account for other agents, and suffer from issues stemming from the environment's non-stationarity induced by other agents learning. Recent methods in multi-agent reinforcement learning [18, 19] attempt to address these issues by leveraging centralized critics in an actor-critic paradigm; however, these methods do not scale well as the number of agents present increases. Our method, multi-actor-attention-critic[20], incorporates attention mechanisms into centralized critics in order to alleviate this issue. Experiments on multi-agent domains demonstrate improved performance and scalability relative to state-of-the-art baselines.

Randomized entity-wise factorization (REFIL) for multi-agent reinforcement learning Over the course of an agent's lifetime it may be required to cooperate with variable teams of agents with differing capabilities/skills; however, common patterns of behavior often emerge among sub-groups of these agents. Our proposed approach, randomized entity-wise factorization for imagined learning (REFIL) [21], attempts to leverage these common patterns to improve generalization across similar teams by randomly factorizing value functions into terms comprised of disjoint sub-groups of entities. By constructing value function predictions in this manner we are better able to predict expected returns in novel combinations of familiar sub-group states. Experiments on complex multi-task multi-agent settings demonstrate improved sample efficiency and generalization over state-of-the-art baselines.

Solving stochastic traveling purchaser problem (STPP) for scavenger hunt service robot Creating robots that can perform general-purpose service tasks in a human-populated environment has been a longstanding grand challenge for AI and Robotics research. One particularly valuable skill that is relevant to a wide variety of tasks is the ability to locate and retrieve objects upon request. In this work, we model this skill as a Scavenger Hunt (SH) game formulated as a variation of the NP-hard stochastic traveling purchaser problem. In this problem, the goal is to find a set of objects as quickly as possible, given probability distributions of where they may be found. We investigate the performance of several solution algorithms for the SH problem, both in simulation and on a real mobile robot. We use Reinforcement Learning (RL) to train an agent to plan a minimal cost path, and show that the RL agent can outperform a range of heuristic algorithms, achieving near optimal performance. In order to stimulate research on this problem, we introduce a publicly available software stack and associated website that enable users to upload scavenger hunts which robots can download, perform, and learn from to continually improve their performance on future hunts.

Model-based Lifelong Reinforcement Learning with Bayesian Exploration (VBLRL) We propose a lifelong RL algorithm that extracts the common structure existing in previously encountered tasks so that the agent can quickly learn the dynamics specific to the new tasks. We consider lifelong RL problems that can be modeled as hidden-parameter MDPs or *HiP-MDPs* [22, 23], where variations among the true task dynamics can be described by a set of hidden parameters. Our algorithm goes further than previous work in both lifelong learning and HiP-MDPs by **1)** Separately modeling epistemic and aleatory uncertainty over different levels of abstraction across the collection of tasks: the uncertainty captured by a world-model distribution describing the probability distribution over tasks, and the uncertainty captured by a task-specific model of the (stochastic) dynamics within a single task. To enable more accurate sequential knowledge transfer, we separate the learning process for these two quantities and maintain a hierarchical Bayesian posterior that approximates them. **2)** Performing Bayesian exploration enabled by the hierarchical posterior: The method lets the agent act optimistically according to models sampled from the posterior, and thus increases sample efficiency.

Details on each of these algorithms are provided in the next section.

3 METHODS, ASSUMPTIONS, AND PROCEDURES

This section describes the methodologies and problem formulations of the lifelong learning research conducted under this project. We first describe the integrated lifelong learning framework developed by our system engineering group, which integrates many of our individual research contributions into reconfigurable modules, followed by a description of a challenging lifelong learning setting which we use to evaluate our integrated framework. We then describe the individual research that led to the modules used in our integrated framework, as well as other work that produced major research contributions for the field of lifelong learning.

3.1 Integrated Lifelong Learning Framework

We developed a modular lifelong learning system capable of performing both classification and reinforcement learning (RL) tasks in realistic service robot settings. Figure 1 summarizes the full system. The core of the system integrates factorized models (DF-CNNs for supervised learning [24] and LPG-FTW for RL [25]) with the perception-action loop of a mobile robot, which we divide into separate classification and RL pipelines. The system includes additional optional modules which can be combined with the core classification and RL pipelines, including a task-agnostic feature meta learning module using META-KFO [26], intrinsic motivation via meta-learned intrinsic reward functions [27], an alternative core RL algorithm based on the A2C algorithm [28], a self-supervised exploration module based on active visual mapping for robot navigation [29], and an MDP-based curriculum learning module [30]. These components can be enabled or disabled depending on the problem domain, and we discuss example configurations developed and evaluated for a visual scavenger hunt application in Sections 3.2 and 4.3.

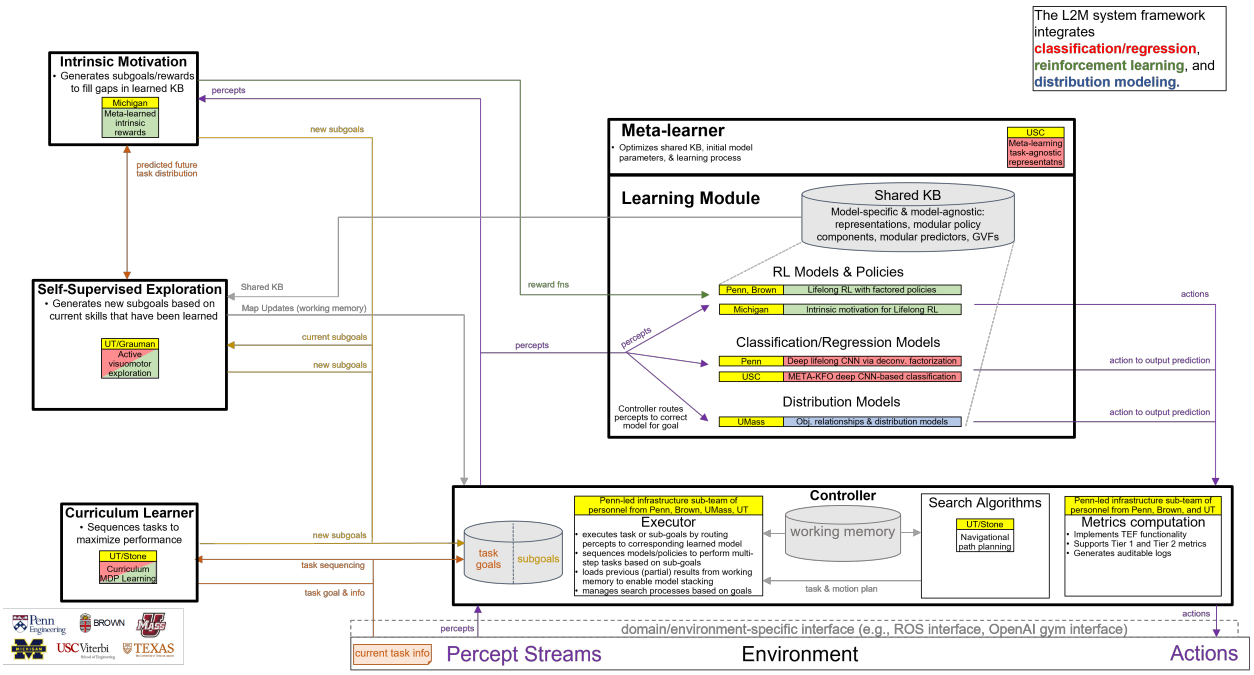


Figure 1: Integrated Lifelong Learning Framework System Architecture

Our integrated lifelong learning framework is developed around a core modular perception-action loop, consisting of the Environment, Controller, and Learning Module system components. An agent receives the current percept from the environment using an environment-specific interface, such as the OpenAI gym interface or a ROS sensor driver. The percept is read by the controller, which passes it on to the task-appropriate learning module. The learning module itself can be one of many approaches for reinforcement learning tasks, classification or regression tasks, or distribution modeling tasks. We give an example of each type of task below.

- RL task: the agent observes the state of the environment for the current time step from a depth camera, which the controller passes as an observation to an actor-critic network for object search problems. The network returns a movement command from the actor network to the controller, which is then executed using an environment-specific interface. This cycle repeats at each time step until the task is complete, with the agent updating its learned policy according to a reward signal that the controller receives from the environment.
- Classification task: the agent observes an image of an unidentified object, which the controller passes to a learning module for an object classification deep neural network, and the model either returns an object label during evaluation/deployment, or updates its model based on the object's ground truth label.
- Distribution query: the agent receives an observation containing the known locations of previously detected objects, which the controller passes to an object co-occurrence matrix to query the possible location of an undetected object. The co-occurrence matrix returns probabilities that the query object will be found in each location of the environment, which the controller then uses to determine future actions for downstream tasks.

This core perception-action loop can be supported by additional modular components, belonging to the following four categories. *Meta-learner* components maintain task-agnostic representations learned over prior tasks that can be used to speed up the learning process for novel tasks, or optimize a learning module's shared knowledge base to better represent the task distribution encountered so far. *Intrinsic Motivation* components generate their own reward signals, such as a reward that encourages exploration, to supplement the extrinsic rewards provided by the environment. *Self-Supervised Exploration* components generate subgoals that are either complimentary to the current task, or expected to be useful for predicted future tasks, which may be more efficient to learn first rather than learning a given task with no prior experience or skills. *Curriculum Learner* components can be used in applications where an agent is given a set of tasks to learn, and determine optimal task orderings to maximize performance in a lifelong learning setting. Depending on which of these additional modules are included in a system configuration, they can interact with each other. For example, a Self-Supervised Exploration component can generate a set of potentially useful subgoals, which can be learned using different reward functions generated by an Intrinsic Motivation component. These new self-generated tasks can in turn be sequenced with a Curriculum Learner module to determine which tasks, if any, would be most useful to learn before training on an extrinsic task given by the problem domain.

To contextualize every component presented in this paper within our integrated lifelong learning framework, we organize each component into its corresponding integrated framework modules

Table 1: Full List of Components and Integration Status

Component	Framework Module	Integration Status
DF-CNN	Learning Module (Classification/Regression)	fully integrated
LPG-FTW	Learning Module (RL)	fully integrated
OBJMAP	Learning Module (Distribution)	fully integrated
KFO	Meta-Learner & Learning Module (Cl./Re.)	fully integrated
LIRPG	Intrinsic Motivation & Learning Module (RL)	fully integrated
OCCANT	Self-Supervised Exploration	fully integrated
CMDP	Curriculum Learner	fully integrated
COMPCLF	Learning Module (Classification/Regression)	not integrated
COMPRL	Learning Module (RL)	not integrated
VBLRL	Meta-Learner & Learning Module (RL)	not integrated
ELIRL	Meta-learner	not integrated
REFIL	Meta-learner (multi-agent extension)	not integrated
SHELS	Self-Supervised Exploration & Learning Module (Cl./Re.)	not integrated
HAL	Self-Supervised Exploration & Learning Module (RL)	not integrated
DETFLICK	Self-Supervised Exploration	not integrated
STSL	Self-Supervised Exploration	not integrated
MAAC	Controller (multi-agent extension)	not integrated
STPP	Controller & Environment	not integrated
HNH	Environment & Learning Module (Distribution)	compatible for eval.
CompoSuite	Environment	compatible for eval.

and provide their current integration status in Table 1. While all of our developed methods are compatible with the L2M Framework we developed, we only integrated those components that were (1) sufficiently mature, and (2) compatible with and necessary for our applications to the APL MiniGrid benchmark and Service Robot Scavenger Hunt. Links to implementations of stand-alone methods are available in Appendix B.

For more information about the algorithms and approaches behind each fully-integrated component in our system, please refer to their corresponding subsections within Section 3.3, specifically:

- Deep lifelong CNN via deconvolutional factorization (DF-CNN), Section 3.3.1
- Lifelong RL with factored policies (LPG-FTW), Section 3.3.2
- META-KFO deep CNN-based classification / Meta-learning task-agnostic representations, Section 3.3.3
- Active visuomotor exploration (OCCANT), Section 3.3.4
- Intrinsic motivation for lifelong RL / Meta-learned intrinsic rewards (LIRPG), Section 3.3.5
- Curriculum MDP learning (CMDP), Section 3.3.6
- Obj. relationships & distribution models (OBJMAP), Section 3.3.7

3.2 Lifelong Robot Learning for the Visual Scavenger Hunt

To provide a specific context for discussing and evaluating our modular integrated lifelong learning framework, we focus on visual scavenger hunt tasks for mobile service robots. We developed lifelong learning scenarios for classification tasks and reinforcement learning tasks by specifying sets of object classification and object search tasks, respectively. These tasks were then ordered into a curriculum of learning blocks, with one task per block, interspersed with evaluation blocks that contained all tasks in the curriculum. The goal of the lifelong learning agent is to efficiently learn each task in the curriculum in sequence, while maintaining performance over all previously learned tasks. We developed the task specification, experimental curricula, and agent using household environments in the AI Habitat simulator [31], constructed using the Matterport 3D data set [32], resulting in realistic observations for household service robots derived from real world sensor data. An example of Matterport 3D data, with associated ground truth furniture segmentation, is shown in Figure 2. Classification and reinforcement learning tasks were specified as follows.

Figure 2: Sample Sensor Data (left) and Ground Truth Mapping (right) from Matterport 3D

Classification Setting Lifelong classification learning curricula were constructed from data sets consisting of rgb camera viewpoints containing positive examples of objects near the agent (i.e. objects are high resolution, and account for a large portion of each image). These viewpoints were collected by simulated agents performing random walks through the simulated household environments, labeled with ground truth data from the Matterport 3D data set. A single task requires a mobile agent to classify a set of objects taken from an object superclass. Each classification task is set up as a standard multi-class supervised learning problem, where the agent must correctly select an object label for a given image, with multi-class classification accuracy as the primary metric. Sample images with labels are given in Figure 3.

Example tasks used to define classification curricula include:

- Classify objects {`chair`, `sofa`, `cushion`, `misc_seating`} from the superclass `seating_furniture`.
- Classify objects {`sink`, `toilet`, `bathtub`, `counter`} from the superclass `plumbing_furniture`.

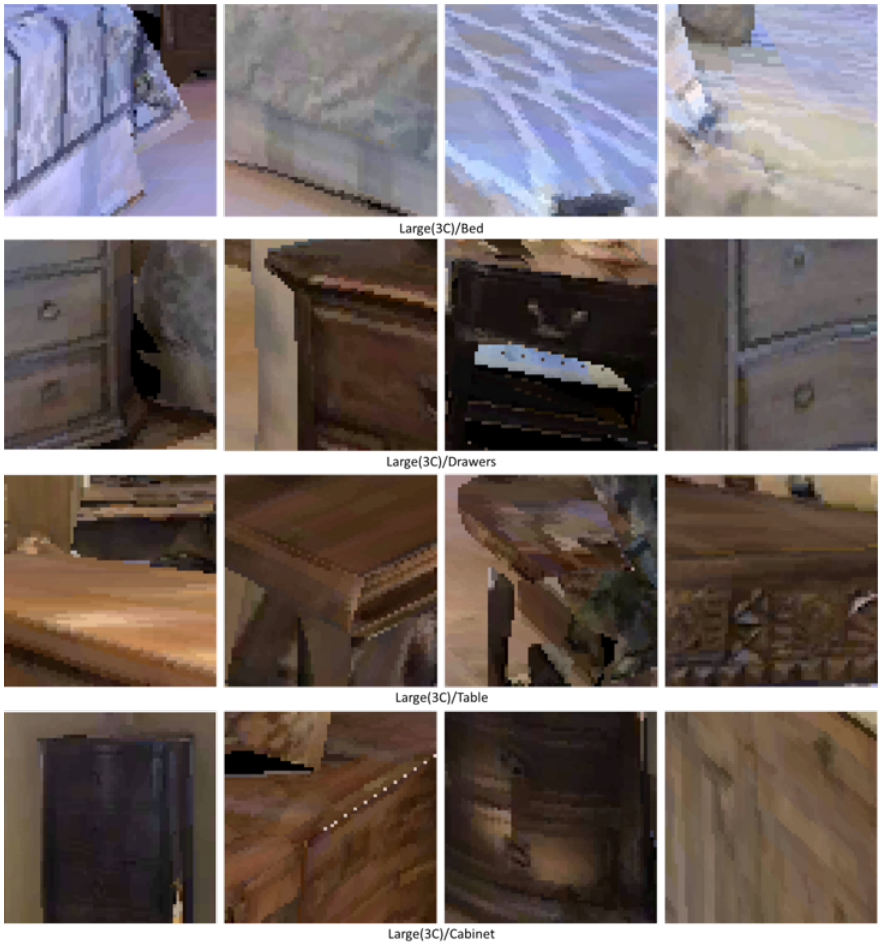


Figure 3: Sample Classification Task Inputs and Labels

- Classify objects {`bed`, `chest_of_drawers`, `table`, `cabinet`} from the superclass `large_furniture`.
- Classify objects {`plant`, `tv_monitor`, `picture`, `clothes`} from the superclass `non-furniture_objects`.

Reinforcement Learning Setting Lifelong reinforcement learning curricula were defined for a mobile agent operating in AI Habitat. A single task requires a mobile agent to find one instance of a given target object type in a given household environment (e.g. an apartment or a town house). Each task is set up as a standard reinforcement learning (RL) problem in which an agent must learn a policy that maps observations to actions to maximize a reward. Observations consisted of the simulated robot's camera frames, actions consisted of simulated control actions {`move_forward`, `turn_left`, `turn_right`}. We defined a sparse positive reward when the agent is within a minimum distance of the target object, and the target object is visible in the agent's viewpoint, discounted by the time it takes the agent to reach the object. A sample path generated by a learned policy and the final observation image for a find `chair` task are given in Figure 4.

We defined one RL task per classification superclass per environment, such as `find_chair` for the `seating_furniture` superclass. Differences in tasks arise naturally from the intrinsic

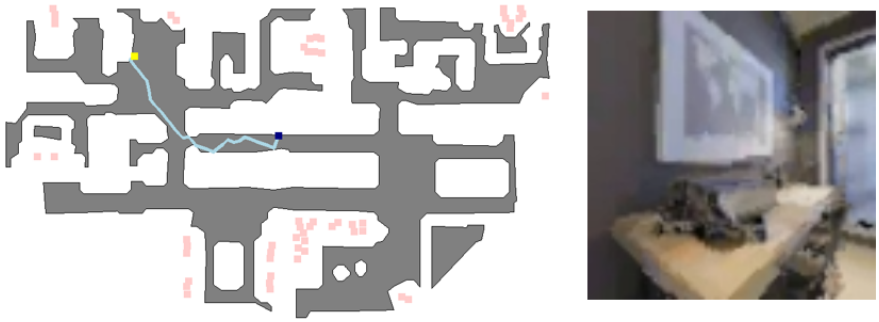


Figure 4: Sample RL Task Result Visualization

characterstics of each environment and target object, such as lighting, sensor noise, number and type of distractor objects, etc. We allowed the agents to use models pre-trained on external data sets, but had no knowledge of the house layouts, decor, or target objects until deployment in the lifelong learning scenario.

3.3 Details on Individual Lifelong Learning Components within the L2M System

In this section, we detail each of the individual components used within our integrated L2M system, as described in Table 1 previously. Please note that these components can each be used as stand-alone algorithms for various aspects of the lifelong learning problem, as described in their respective subsections. Corresponding evaluation details and results for each approach are provided in Section 4.4. Links to implementations are available in Appendix B.

3.3.1 Deconvolutional factorized CNN (DF-CNN) for lifelong deep learning.

Lifelong learning of classification tasks A lifelong learning system faces a series of consecutive tasks $\mathcal{Z}_1, \ldots, \mathcal{Z}_{T_{\max}}$, and must learn a model (i.e., a classifier) for each task. Here, we assume that the system has no *a priori* information about tasks such as the task distribution, order, or total number of tasks $T_{\max}$. For the classification setting, each task $\mathcal{Z}_t$ is represented by a pair of an associated data feature space $\mathcal{X}_t$ and label space $\mathcal{Y}_t$, from which labeled examples are drawn. Each task $\mathcal{Z}_t$ admits an associated convolutional neural network $CNN_t : \mathcal{X}_t \mapsto \mathcal{Y}_t$, with d layers, trained on labeled data for that task. In lifelong learning framework, each CNN_t is trained consecutively, possibly with transfer from any previously learned tasks $\mathcal{Z}_1, \ldots, \mathcal{Z}_{t-1}$.

Factorized transfer To enable transfer among the different CNN_t task models, we draw inspiration from the use of factorized transfer in shallow MTL [33, 34] and lifelong learning [35] methods. These shallow methods learn a set of T task-specific linear models parameterized by $\boldsymbol{W} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T] \in \mathbb{R}^{d \times T}$ and assume that these model parameters admit a rank-constrained matrix factorization of the form $\boldsymbol{W} = \boldsymbol{L}\boldsymbol{S}$, where $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ is a basis over the model parameter space, $\boldsymbol{S} \in \mathbb{R}^{k \times T}$ are the coefficients over this basis to reconstruct the parameters, and k is the dimension of the latent space. In effect, these approaches learn a knowledge base $\boldsymbol{L}$ that represents a shared subspace for the model parameters, and facilitate transfer to new tasks by learning models within this subspace.

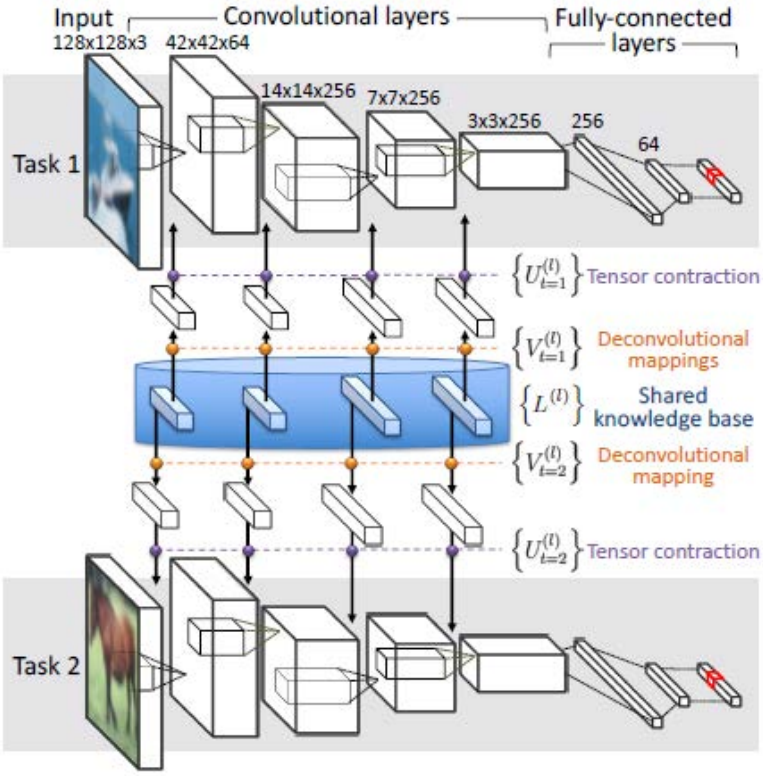


Figure 5: The DF-CNN Architecture Depicted for Two Tasks

Since these methods only operate on linear task models, we cannot adopt them directly for use on deep nets. However, we show next that we can adapt this notion of a shared knowledge base in combination with a novel type of factorized transfer via deconvolution to operate effectively on CNNs.

Deconvolutional factorized CNN Our architecture, called a *deconvolutional factorized CNN* (DF-CNN), seeks to address this lifelong learning problem using deep convolutional networks with a shared knowledge base to enable transfer between tasks (Fig. 5). To facilitate transfer, our architecture maintains a shared latent knowledge base connecting the various layers *across* the task-specific CNNs. The filters of the CNNs are generated from the learned latent knowledge base by the deconvolution operator (transposed convolution), followed by a tensor contraction. Unlike previous methods that involve tensor factorization to achieve sparsity, our proposal is naturally sparse by virtue of the deconvolution operator.

For each convolutional layer $l \in \{1, \ldots, d\}$ of each task-specific CNN_t, let $W_t^{(l)} \in \mathbb{R}^{h \times w \times c_{\text{in}} \times c_{\text{out}}}$ denote its corresponding filters where h and w are the filter height and width, and c_{in} and c_{out} are the numbers of input and output channels.

To enable transfer between the convolutional layers of different CNN_t task models, we introduce a task-*independent* layer-*dependent* shared knowledge base $L^{(l)}$ for each layer l, which is shared across all tasks. Following similar assumptions used in factorized transfer for shallow models, we assume that each $W^{(l)}$ is derived from the corresponding shared latent knowledge base $L^{(l)}$, enabling connections between filters at the l-th layer of different task models.

Algorithm 1 DF-CNN (λ, *kbSize*, *transformSize*)

$L^{(1:d)} \leftarrow \text{randInit}(kbSize)$
while isAnotherTaskAvailable() **do**
  $(X_t, y_t, t) \leftarrow \text{getNextTaskTrainingData}()$
  if $\text{isNewTask}(t)$ **then**
    $\left(V_t^{(1:d)}, U_t^{(1:d)}\right) \leftarrow \text{randomInit}(transformSize)$
  while continueBatchTraining() **do**
    // reconstruct NN from shared KB
    for $l = 1$ to d **do**
      $D_t^{(l)} \leftarrow \text{deconv}(L^{(l)}, V_t^{(l)})$
      $W_t^{(l)} \leftarrow \text{tensorDot}(D_t^{(l)}, U_t^{(l)})$
    $taskNet_t \leftarrow \text{buildNeuralNet}\left(W_t^{(1:d)}\right)$
    // update shared KB and task-specific transforms
    $X_b, y_b \leftarrow \text{drawMiniBatch}(X_t, y_t)$
    $\left(L^{(1:d)}, V_t^{(1:d)}, U_t^{(1:d)}\right) \leftarrow \text{gradientOptimizer}(X_b, y_b, taskNet_t, \lambda)$

Specifically, we utilize a deconvolutional mapping and a tensor contraction to extract abstract knowledge from the filters $\{W_t^{(l)}\}_{t=1}^{T}$ into the shared knowledge base $L^{(l)}$, which we take to be a 3rd-order tensor $L^{(l)} \in \mathbb{R}^{\hat{h}\times\hat{w}\times\hat{c}}$. We first deconvolve $L^{(l)}$ into

$$D_t^{(l)} = \text{deconv}(L^{(l)}; V_t^{(l)}) \ , \tag{1}$$

where $D_t^{(l)}$ is a 3rd-order tensor of size $h \times w \times c$, $V_t^{(l)} \in \mathbb{R}^{p\times p\times\hat{c}\times c}$ is the filter of the task-dependent deconvolutional mapping, and p is the spatial size of the deconvolutional filters. The deconvolutional mapping learns to generate a basis of convolutional filters within $L^{(l)}$. We then apply the tensor contraction to reconstruct each $W_t^{(l)}$ based on $D_t^{(l)}$:

$$W_t^{(l)} = D_t^{(l)} \bullet U_t^{(l)} = \sum_{k=1}^{c} D_{t,(\cdot,\cdot,k)}^{(l)} U_{t,(k,\cdot,\cdot)}^{(l)} \ , \tag{2}$$

where $U_t^{(l)}$ is a 3rd-order tensor of size $c \times c_{\text{in}} \times c_{\text{out}}$, and both subscripts $(k, \cdot, \cdot)$ and $(\cdot, \cdot, k)$ express the elements' index in the tensor. Similar to channel-wise convolution, the tensor contraction expresses the filter as a linear combination of the basis vectors, transforming $D_t^{(l)}$ to reconstruct the convolution filter $W_t^{(l)}$ by changing the size of channels. Note that $V_t^{(l)}$ and $U_t^{(l)}$ are task-specific, and serve to transform the shared knowledge bases into a model specific to task $\mathcal{Z}_t$.

Similar to the work of dynamic filter generation [36, 37], a single operation can be employed to expand the knowledge base into a large task-specific filter. However, in our proposal, deconvolution and tensor contraction are used instead as a two-staged expansion, distinguishing between the transfer process along the spatial axis of the images and along the channels of the images.

Training the DF-CNN architecture Our learning approach must update both the shared knowledge bases and task-specific knowledge transformations while training on each task in a lifelong setting. The knowledge bases $\{L^{(l)}\}_{l=1}^{d}$ and task-specific knowledge transformations $\{(V_t^{(l)}, U_t^{(l)})\}_{l=1}^{d}$

can be trained end-to-end via gradient-based optimization. The process of training the DF-CNN is described in Algorithm 1. As hyperparameters, the algorithm requires the learning rate λ of the optimizer, the size of the knowledge bases ($kbSize \in \mathbb{R}^{3d}$), and the dimensions of the task-specific tensors $V_t^{(l)}$ and $U_t^{(l)}$ ($transformSize \in \mathbb{R}^{(4d+3d)}$).

The shared knowledge bases $\{L^{(l)}\}_{l=1}^{d}$ are randomly initialized prior to learning the first task. Upon receiving the labeled training data for each new task $\mathcal{Z}_t$, the task-specific knowledge transformations $\{(V_t^{(l)}, U_t^{(l)})\}_{l=1}^{d}$ are first initialized randomly, and then these parameters along with the knowledge bases $\{L^{(l)}\}_{l=1}^{d}$ are updated according to the observed training instances (X_t, y_t). During training on task $\mathcal{Z}_t$, the knowledge transformations for all tasks except t are held unchanged, while the shared knowledge bases are updated. Since the convolutional filters for each CNN_t are generated dynamically from the shared knowledge bases, changes to $\{L^{(l)}\}_{l=1}^{d}$ can affect previously trained networks $CNN_1, \ldots, CNN_{t-1}$ *without* retraining those networks; this phenomenon is known as reverse transfer [35]. Despite the lack of explicit mechanisms to prevent catastrophic forgetting [38] (i.e., severe negative reverse transfer) in these previously learned models, we show empirically that deconvolutional factorization of the model parameter space resists catastrophic forgetting and indeed exhibits positive reverse transfer—these results mirror similar results on lifelong learning of shallow linear models via factorized transfer [35].

3.3.2 Lifelong Policy Gradients: Faster Training Without Forgetting (LPG-FTW).

RL via policy gradients In a Markov decision process (MDP) $\langle \mathcal{X}, \mathcal{U}, T, R, \gamma \rangle$, $\mathcal{X} \subseteq \mathbb{R}^d$ is the set of states, $\mathcal{U} \subseteq \mathbb{R}^m$ is the set of actions, $T : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mapsto [0, 1]$ is the probability $P(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u})$ of going to state $\boldsymbol{x}'$ after executing action $\boldsymbol{u}$ in state $\boldsymbol{x}$, $R : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is the reward function measuring the goodness of a state-action pair, and $\gamma \in [0, 1)$ is the discount factor for future rewards. A policy $\pi : \mathcal{X} \times \mathcal{U} \mapsto [0, 1]$ prescribes the agent's behavior as a probability $P(\boldsymbol{u} \mid \boldsymbol{x})$ of selecting action $\boldsymbol{u}$ in state $\boldsymbol{x}$. The goal of RL is to find the policy π^* that maximizes the expected returns $\mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_i\right]$, where $R_i = R(\boldsymbol{x}_i, \boldsymbol{u}_i)$.

PG algorithms have shown success in solving continuous RL problems by assuming that the policy $\pi_{\boldsymbol{\theta}}$ is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$ and searching for the set of parameters $\boldsymbol{\theta}^*$ that optimizes the long-term rewards: $\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_i\right]$ [39, 40, 41]. Different approaches use varied strategies for estimating the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$. However, the common high-level idea is to use the current policy $\pi_{\boldsymbol{\theta}}$ to sample trajectories of interaction with the environment, and then estimating the gradient as the average of some function of the features and rewards encountered through the trajectories.

The lifelong learning problem We frame lifelong PG learning as online multi-task learning of policy parameters. The agent will face a sequence of tasks $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T_{\max})}$, each of which will be an MDP $\mathcal{Z}^{(t)} = \langle \mathcal{X}^{(t)}, \mathcal{U}^{(t)}, T^{(t)}, R^{(t)}, \gamma \rangle$. Tasks are drawn *i.i.d.* from a fixed, stationary environment. The goal of the agent is to find the policy parameters $\left\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T_{\max})}\right\}$ that maximize the performance across all tasks: $\frac{1}{T_{\max}} \sum_{t=1}^{T_{\max}} \mathbb{E} \sum_{i=0}^{\infty} \gamma^i R_i^{(t)}$. We do not assume knowledge of the total number of tasks, the order in which tasks will arrive, or the relations between different tasks.

Upon observing each task, the agent will be allowed to interact with the environment for a limited time, typically insufficient for obtaining optimal performance without exploiting information from prior tasks. During this time, the learner will strive to discover any relevant information from the current task to 1) relate it to previously stored knowledge in order to permit transfer and 2) store

Algorithm 2 LPG-FTW(d, k, λ, μ, M)

$T \leftarrow 0, \quad \boldsymbol{L} \leftarrow \text{initializeL}(d, k)$
loop
  $t \leftarrow \text{getTask}()$
  if $\text{isNewTask}(t)$ **then**
    $\boldsymbol{s}^{(t)} \leftarrow \text{initializeSt}(k)$
    $T \leftarrow T + 1$
  else
    $\boldsymbol{A} \leftarrow \boldsymbol{A} - 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
    $\boldsymbol{b} \leftarrow \boldsymbol{b} - \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$
  for $i = 1, \ldots, N$ **do**
    $\mathbb{T} \leftarrow \text{getTrajectories}(\boldsymbol{L}\boldsymbol{s}^{(t)})$
    $\boldsymbol{s}^{(t)} \leftarrow \text{PGStep}(\mathbb{T}, \boldsymbol{L}, \boldsymbol{s}^{(t)}, \mu)$
    if $i \mod M = 0$ **then**
      $\boldsymbol{\alpha}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
      $\boldsymbol{g}^{(t)}, \boldsymbol{H}^{(t)} \leftarrow \text{gradAndHess}(\boldsymbol{\alpha}^{(t)})$
      $\boldsymbol{A}_{\mathsf{tmp}} \leftarrow \boldsymbol{A} + 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
      $\boldsymbol{b}_{\mathsf{tmp}} \leftarrow \boldsymbol{b} + \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$
      $\text{vec}(\boldsymbol{L}) \leftarrow \left(\frac{1}{T}\boldsymbol{A}_{\mathsf{tmp}} - 2\lambda\boldsymbol{I}\right)^{-1}\left(\frac{1}{T}\boldsymbol{b}_{\mathsf{tmp}}\right)$
  $\boldsymbol{A} \leftarrow \boldsymbol{A}_{\mathsf{tmp}}, \quad \boldsymbol{b} \leftarrow \boldsymbol{b}_{\mathsf{tmp}}$

any newly discovered knowledge for future reuse. At any time, the agent may be evaluated on any previously seen task, so it must retain knowledge from all early tasks in order to perform well.

Lifelong policy gradient learning Our framework for lifelong PG learning uses factored representations. The central idea is assuming that the policy parameters for task t can be factored into $\boldsymbol{\theta}^{(t)} \approx \boldsymbol{L}\boldsymbol{s}^{(t)}$, where $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ is a shared dictionary of policy factors and $\boldsymbol{s}^{(t)} \in \mathbb{R}^k$ are task-specific coefficients that select components for the current task. We further assume that we have access to some base PG algorithm that, given a single task, is capable of finding a parametric policy that performs well on the task, although not necessarily optimally.

Upon encountering a new task $\mathcal{Z}^{(t)}$, LPG-FTW (Algorithm 2) will use the base learner to optimize the task-specific coefficients $\boldsymbol{s}^{(t)}$, without modifying the knowledge base $\boldsymbol{L}$. This corresponds to searching for the optimal policy that can be obtained by combining the factors of $\boldsymbol{L}$. Every $M \gg 1$ steps, the agent will update the knowledge base $\boldsymbol{L}$ with any relevant information collected from $\mathcal{Z}^{(t)}$ up to that point. This allows the agent to search for policies with an improved knowledge base in subsequent steps.

Concretely, the agent will strive to solve the following optimization during the training phase:

$$\boldsymbol{s}^{(t)} = \arg\max_{\boldsymbol{s}} \ell(\boldsymbol{L}_{t-1}, \boldsymbol{s}) = \arg\max_{\boldsymbol{s}} \mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s}) - \mu\|\boldsymbol{s}\|_1 \ , \tag{3}$$

where $\boldsymbol{L}_{t-1}$ denotes the $\boldsymbol{L}$ trained up to task $\mathcal{Z}^{(t-1)}$, $\mathcal{J}^{(t)}(\cdot)$ is any PG objective, and the ℓ_1 norm encourages sparsity. Following [42], the agent will then solve the following second-order approxi-

mation to the multi-task objective to add knowledge from task $\mathcal{Z}^{(t)}$ into the dictionary:

$$\begin{aligned}\boldsymbol{L}_t = \arg\max_{\boldsymbol{L}} \hat{g}_t(\boldsymbol{L}) = \arg\max_{\boldsymbol{L}} -\lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 \\ + \frac{1}{t}\sum_{\hat{t}=1}^{t} \hat{\ell}(\boldsymbol{L}, \boldsymbol{s}^{(\hat{t})}, \boldsymbol{\alpha}^{(\hat{t})}, \boldsymbol{H}^{(\hat{t})}, \boldsymbol{g}^{(\hat{t})}) \ ,\end{aligned} \tag{4}$$

where $\hat{\ell}(\boldsymbol{L}, \boldsymbol{s}^{(\hat{t})}, \boldsymbol{\alpha}^{(\hat{t})}, \boldsymbol{H}^{(\hat{t})}, \boldsymbol{g}^{(\hat{t})}) = -\mu\|\boldsymbol{s}^{(\hat{t})}\|_1 + \|\boldsymbol{\alpha}^{(\hat{t})} - \boldsymbol{L}\boldsymbol{s}^{(\hat{t})}\|^2_{\boldsymbol{H}^{(\hat{t})}} + \boldsymbol{g}^{(\hat{t})\top}(\boldsymbol{L}\boldsymbol{s}^{(\hat{t})} - \boldsymbol{\alpha}^{(\hat{t})})$ is the second-order approximation to the objective of a previously seen task, $\mathcal{Z}^{(\hat{t})}$. The objective function's gradient $\boldsymbol{g}^{(\hat{t})} = \nabla_{\boldsymbol{\theta}}\mathcal{J}^{(\hat{t})}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\alpha}^{(\hat{t})}}$ and Hessian $\boldsymbol{H}^{(\hat{t})} = \frac{1}{2}\nabla_{\boldsymbol{\theta},\boldsymbol{\theta}^\top}\mathcal{J}^{(\hat{t})}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\alpha}^{(\hat{t})}}$ are evaluated at the policy for task $\mathcal{Z}^{(\hat{t})}$ immediately after training, $\boldsymbol{\alpha}^{(\hat{t})} = \boldsymbol{L}_{\hat{t}-1}\boldsymbol{s}^{(\hat{t})}$. The solution to this optimization can be obtained in closed form as $\mathrm{vec}(\boldsymbol{L}_t) = \boldsymbol{A}^{-1}\boldsymbol{b}$, where the two components $\boldsymbol{A} = -2\lambda\boldsymbol{I} + \frac{2}{t}\sum_{\hat{t}=1}^{t}(\boldsymbol{s}^{(\hat{t})}\boldsymbol{s}^{(\hat{t})\top}) \otimes \boldsymbol{H}^{(\hat{t})}$ and $\boldsymbol{b} = \frac{1}{t}\sum_{\hat{t}=1}^{t}\boldsymbol{s}^{(\hat{t})} \otimes (-\boldsymbol{g}^{(\hat{t})} + 2\boldsymbol{H}^{(\hat{t})}\boldsymbol{\alpha}^{(\hat{t})})$. Notably, these can be computed incrementally as each new task arrives, so that $\boldsymbol{L}$ can be updated without preserving data or parameters from earlier tasks. Moreover, the Hessians $\boldsymbol{H}^{(\hat{t})}$ needed to compute $\boldsymbol{A}$ and $\boldsymbol{b}$ can be discarded after each task if the agent is not expected to revisit tasks for further training. If instead the agent will revisit tasks multiple times (e.g., for interleaved multi-task learning), then each $\boldsymbol{H}^{(\hat{t})}$ must be stored at a cost of $O(d^2 T_{\mathsf{max}})$.

Intuitively, in Equation 3 the agent leverages knowledge from all past tasks while training on task $\mathcal{Z}^{(t)}$, by searching for $\boldsymbol{\theta}^{(t)}$ in the span of $\boldsymbol{L}_{t-1}$. This makes LPG-FTW fundamentally different from prior multi-model methods that learn each task's parameter vector in isolation and subsequently combine prior knowledge to improve performance. One potential drawback is that, by restricting the search to the span of $\boldsymbol{L}_{t-1}$, we might miss other, potentially better, policies. However, any set of parameters far from the space spanned by $\boldsymbol{L}_{t-1}$ would be uninformative for the multi-task objective, since the approximations to the previous tasks would be poor near the current task's parameters and vice versa. In Equation 4, LPG-FTW approximates the loss around the current set of parameters $\boldsymbol{\alpha}^{(t)}$ via a second-order expansion and finds the $\boldsymbol{L}_t$ that optimizes the average approximate cost over all previously seen tasks, ensuring that the agent does not forget the knowledge required to solve them.

Knowledge base initialization The intuition we have built holds only when a reasonably good $\boldsymbol{L}$ matrix has already been learned. But what happens at the beginning of the learning process, when the agent has not yet seen a substantial number of tasks? If we take the naïve approach of initializing $\boldsymbol{L}$ at random, then the $\boldsymbol{s}^{(t)}$'s are unlikely to be able to find a well-performing policy, and so updates to $\boldsymbol{L}$ will not leverage any useful information.

One common alternative is to initialize the k columns of $\boldsymbol{L}$ with the STL solutions to the first k tasks. However, this method prevents tasks $\mathcal{Z}^{(2)}, \ldots, \mathcal{Z}^{(k)}$ from leveraging information from earlier tasks, impeding them from achieving potentially higher performance. Moreover, several tasks might rediscover information, leading to wasted training time and capacity of $\boldsymbol{L}$.

We propose an initialization method (Algorithm 3) that enables early tasks to leverage knowledge from previous tasks and prevents the discovery of redundant information. The algorithm starts from an empty dictionary and adds error vectors $\boldsymbol{\epsilon}^{(t)} \in \mathbb{R}^d$ for the initial k tasks. For each task $\mathcal{Z}^{(t)}$, we modify Equation 3 for learning $\boldsymbol{s}^{(t)}$ by adding $\boldsymbol{\epsilon}^{(t)}$ as additional learnable parameters:

$$\boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)} = \arg\max_{\boldsymbol{s},\boldsymbol{\epsilon}} \mathcal{J}^{(t)}(\boldsymbol{L}_{t-1}\boldsymbol{s} + \boldsymbol{\epsilon}) - \mu\|\boldsymbol{s}\|_1 - \lambda\|\boldsymbol{\epsilon}\|_2^2 \ .$$

Algorithm 3 InitializeL(d, k, λ, μ)

$T \leftarrow 0, \quad \boldsymbol{L} \leftarrow$empty$(d, 0)$
while $T < k$ **do**
  $t \leftarrow$getTask()
  $\boldsymbol{s}^{(t)} \leftarrow$initializeSt($k$)
  $T \leftarrow T + 1$
  for $i = 1, \ldots, N$ **do**
    $\mathbb{T} \leftarrow$ getTrajectories($\boldsymbol{L}\boldsymbol{s}^{(t)}$)
    $\boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)} \leftarrow$ PGStep($\mathbb{T}, \boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\epsilon}^{(t)}, \mu$)
  $\boldsymbol{L} \leftarrow$ addColumn($\boldsymbol{L}, \boldsymbol{\epsilon}^{(t)}$)
  $\boldsymbol{\alpha}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)} + \boldsymbol{\epsilon}^{(t)}$
  $\boldsymbol{g}^{(t)}, \boldsymbol{H}^{(t)} \leftarrow$ gradAndHess($\boldsymbol{\alpha}^{(t)}$)
  $\boldsymbol{A} \leftarrow \boldsymbol{A} + 2\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{H}^{(t)}$
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{s}^{(t)} \otimes \left(-\boldsymbol{g}^{(t)} + 2\boldsymbol{H}^{(t)}\boldsymbol{\alpha}^{(t)}\right)$

Each $\boldsymbol{\epsilon}^{(t)}$ finds knowledge of task $\mathcal{Z}^{(t)}$ not contained in $\boldsymbol{L}$ and is then incorporated as a new column of $\boldsymbol{L}$. Note that this initialization process replaces the standard PG training of tasks $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(k)}$, and therefore does not require collecting additional data beyond that required by the base PG method.

3.3.3 Meta-Optimizer for Fast Adaptation (KFO).

Main idea In the following we give a brief account of various approaches for learning to optimize the lifelong learner, using our META-KFO method. META-KFO is able to merely transform the gradients of a smaller model without increasing its modeling capacity but still results in better meta-learnability. Furthermore, the improvement diminishes when the smaller model gets bigger. We surmise why sufficiently large deep models can meta-learn: *the upper layers have the equivalent effect of transforming the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed only of the bottom layers*.

META-KFO and other meta-optimizer/preconditioning methods A meta-optimizer is a parameterized function U_ξ defining the model's parameter updates. For example, a linear meta-optimizer might be defined as:

$$U_\xi(g) = Ag + b, \tag{5}$$

where $\xi = (A, b)$ is the set of parameters of the linear transformation. The objective is to jointly learn the model and optimizer's parameters ξ to accelerate optimization. Motivated by the analysis of meta-learning in deep nets, we propose to use such an optimizer to transform the gradient updates:

$$\mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta) = \mathbb{E}_{\tau \sim p(\tau)}[\ell_\tau\left(\theta - \alpha U_\xi(\nabla \ell_\tau(\theta))\right)] \tag{6}$$

that takes a similar role of the upper-layers in deep nets in minimizing the MAML loss:

$$\theta \leftarrow \theta - \beta \frac{\partial \mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta)}{\partial \theta}, \qquad \xi \leftarrow \xi - \beta \frac{\partial \mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta)}{\xi} \tag{7}$$

where β is the meta-update learning rate. In this notation, Meta-Curvature [43] implements

$$\text{(MC)} \quad U_\xi\left(\nabla \ell\left(\theta\right)\right) = M \nabla \ell\left(\theta\right) \tag{8}$$

where M is a matrix (block-diagonal tensor factorized). When M is diagonal, this becomes the Meta-SGD [44]. Furthermore, when M is identity, this become MAML. For T-Nets (to be used with MAML-loss), the model parameters are expanded with affine transformations,

$$\text{(T-NETS)} \quad \ell_\tau\left(\mathcal{A}(\theta) - \alpha \nabla \ell_\tau(\mathcal{A}(\theta))\right)) \tag{9}$$

where the transformation $\mathcal{A}(\cdot)$ contains two components $(\mathcal{W}, \mathcal{T})$. $\mathcal{T}$ is shared by all the tasks and $\mathcal{W}$ is task-specific. Since $\mathcal{A}$ is linear, it can be absorbed into the original model after adaptation. For WarpGrad [45], the transformation $\mathcal{A}$ is defined with nonlinear layers, thus strictly increasing the size of the original model (thus, is not considered in this work).

Meta Kronecker Factorized Optimizer Let $g \in \mathbb{R}^k$ be the vectorized gradient $G \in \mathbb{R}^{n \times m}$, such that $\text{vec}(G) = g$ and $m \cdot n = k$. Let $A = R^\top \otimes L$ be the Kronecker product of small matrices $R \in \mathbb{R}^{m \times m}$ and $L \in \mathbb{R}^{n \times n}$, we parameterize the linear map on g as [46, Section 9.1]:

$$U_\xi(g) = \text{vec}\left(L \cdot G \cdot R\right) + b, \tag{10}$$

where $b \in \mathbb{R}^k$ and $\xi = (L, R, b)$. By adjusting m and n, the above factorization requires $\mathcal{O}(\max(m, n)^2)$ memory as opposed to $\mathcal{O}(k^2)$ for the non-factored case. Similarly, the computational complexity for the matrix-product can also be reduced.

This linear transformation can be used as a building block to implement more expressive and non-linear meta-optimizers by interleaving nonlinear activation function $\sigma(\cdot)$:

$$\begin{aligned} U_\xi(g) &= U_{W_h} \circ \cdots \circ \sigma \circ U_{W_1} \circ g & (11) \\ &= \text{vec}\left(L_h \sigma\left(\ldots \sigma\left(L_1 \cdot G \cdot R_1\right) \ldots\right) R_h\right), & (12) \end{aligned}$$

where $\xi = (W_1, \ldots, W_h)$ and W_i defines layer-wise linear mapping. Such a scheme can be used to implement arbitrary meta-optimizers – such as convolutional or recurrent ones – so long as the architecture involves a composition of linear maps.

3.3.4 Occupancy Anticipation for Efficient Exploration and Navigation (OCCANT).

In order to navigate visual environments in lifelong settings (as needed for the Scavenger Hunt), we developed an occupancy anticipation approach for efficient exploration and navigation. Our model anticipates areas not directly visible to the agent because of occlusion (e.g., behind a table, around a corner) or due to being outside its FoV. The agent's first-person view is provided in the form of RGB-D images (see Fig. 6 left). The goal is to anticipate the occupancy for a fixed region in front of the agent, and integrate those predictions over time as the agent moves about.

Next, we define the task setup and notation, followed by our approach for occupancy anticipation and a new formulation for exploration that rewards correctly anticipated regions. Then, we explain how our occupancy anticipation model can be integrated into a state-of-the-art approach [47] for autonomous exploration and navigation in 3D environments.

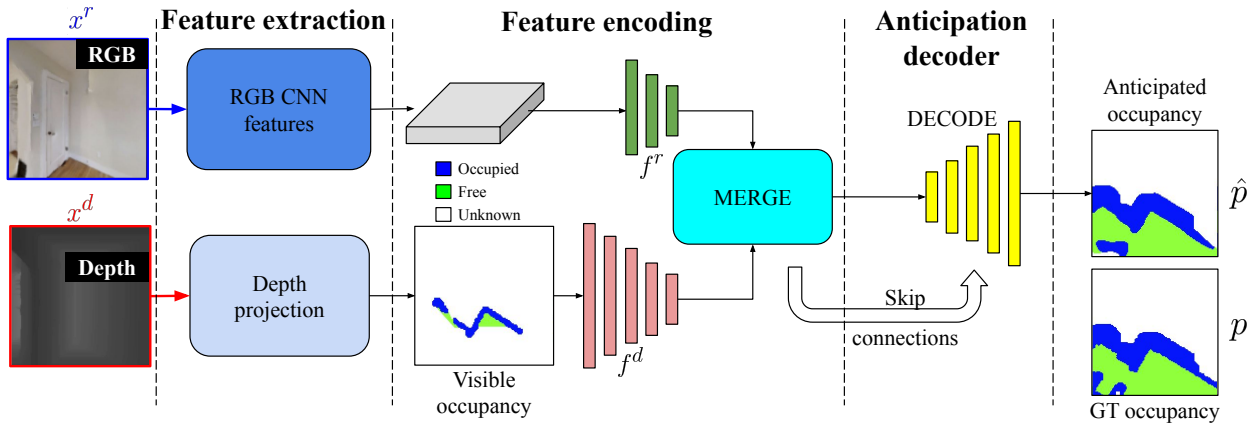


Figure 6: Occupancy Anticipation Model

Occupancy anticipation model We formulate occupancy anticipation as a pixelwise classification task. The egocentric occupancy is represented as a two-channel top-down map $p \in [0,1]^{2\times V\times V}$ which comprises a local area of $V \times V$ cells in front of the camera. Each cell in the map represents a $25\text{mm} \times 25\text{mm}$ region. The two channels contain the probabilities (confidence values) of the cell being occupied and explored, respectively. A cell is considered to be occupied if there is an obstacle, and it is explored if we know whether it is occupied or free. For training, we use the 3D meshes of indoor environments (Sec. 4.4.4) to obtain the ground-truth local occupancy of a $V \times V$ region in front of the camera, which includes parts that may be occluded or outside the field of view (Fig. 6, bottom right).

Our occupancy anticipation model consists of three main components (Fig. 6):

(1) Feature extraction: Given egocentric RGB-D inputs, we compute:

RGB CNN features: We encode the RGB images using blocks 1 and 2 of a ResNet-18 that is pre-trained on ImageNet, followed by three additional convolution layers that prepare these features to be passed forward with the visible occupancy map. This step extracts a mixture of textural and semantic features.

Depth projection: We estimate a map of occupied, free, and unknown space by setting height thresholds on the point cloud obtained from depth and camera intrinsics [48]. Consistent with past work [48, 47], we restrict the projection-based estimates to points within $\sim 3\text{m}$, the range at which modern depth sensors would provide reliable results. This yields the initial visible occupancy map.

(2) Feature encoding: Given the RGB-D features, we independently encode them using UNet [49] encoders and project them to a common feature space. We encode the depth projection features using a stack of five convolutional blocks which results in features $\boldsymbol{f}^d = f^d_{1:5}$. Since the RGB features are already at a lower resolution, we use only three convolutional blocks to encode them, which results in features $\boldsymbol{f}^r = f^r_{3:5}$. We then combine these features using the MERGE module which contains layer-specific convolution blocks to merge each $[\boldsymbol{f}^r_i, \boldsymbol{f}^d_i]$:

$$\boldsymbol{f} = \text{MERGE}(\boldsymbol{f}^d, \boldsymbol{f}^r). \tag{13}$$

For experiments with only the depth modality, we skip the RGB feature extractor and MERGE layer and directly use the occupancy features obtained from the depth image. For experiments with only the RGB modality, we learn a model to infer the visible occupancy features from RGB (to be defined in Sec. 4.4.4) and use that instead of the features computed from the depth image.

(3) Anticipation decoding: Given the encoded features $\boldsymbol{f}$, we use a UNet decoder that outputs a $2 \times V \times V$ tensor of probabilities:

$$\hat{p} = \sigma(\text{DECODE}(\boldsymbol{f})), \tag{14}$$

where $\hat{p} \in [0,1]^{2\times V\times V}$ is the estimated egocentric occupancy and σ is the sigmoid activation function. For training the occupancy anticipation model, we use binary cross entropy loss per pixel and per channel:

$$L = \sum_{i=1}^{V^2}\sum_{j=1}^{2} -\Big[p_{ij}\log \hat{p}_{ij} + (1-p_{ij})\log(1-\hat{p}_{ij})\Big], \tag{15}$$

where p is the ground-truth (GT) occupancy map that is derived from the 3D mesh of training environments.

So far, we have presented our occupancy anticipation approach supposing a single RGB-D observation as input. However, our model is ultimately used in the context of an embodied agent that moves in the environment and actively collects a sequence of RGB-D views to build a complete map of the environment. Next, we introduce a new reward function that utilizes the agent's anticipation performance to guide its exploration during training.

Anticipation reward for exploration policy learning In *visual exploration*, an agent must quickly map a new environment without having a specified target. Prior work on exploration [48, 50, 47, 51] often uses area-coverage—the area seen in the environment during navigation—as a reward function to guide exploration. However, the traditional area-coverage approach is limited to rewarding the agent only for *directly seeing* areas. Arguably, an ideal exploration agent would obtain an accurate and complete map of the environment *without* necessarily directly observing all areas.

Thus, we propose to encourage exploratory behaviors that yield a correctly *anticipated* map. In this case, the occupancy entries in the map need not be obtained via direct agent observations to register a reward; it is sufficient to correctly infer them. In particular, we reward agent actions that yield accurate occupancy predictions for the global environment map, i.e., the number of grid cells where the predicted occupancy matches the layout of the environment.

More concretely, let $\hat{m}_t \in [0,1]^{2\times G\times G}$ be the global environment map obtained by anticipating occupancy for the RGB-D observations $\{x^r_{1:t}, x^d_{1:t}\}$ from time 1 to t, and then geometrically registering the predictions to a single global map based on the agent's pose estimates at each time step (see Fig. 7). Note $G > V$. Let m be the ground-truth layout of the environment. Then, the unnormalized accuracy of a map prediction $\hat{m}$ is measured as follows:

$$\text{Accuracy}(\hat{m}, m) = \sum_{i=1}^{G^2}\sum_{j=1}^{2} \mathbb{1}[\hat{m}_{ij} = m_{ij}], \tag{16}$$

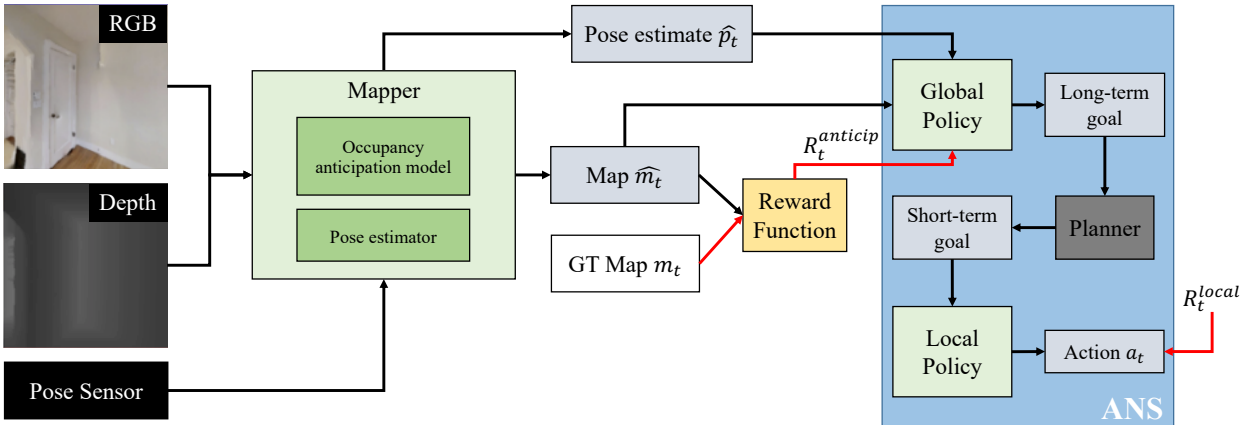


Figure 7: Exploration with Occupancy Anticipation

where $\mathbb{1}[\hat{m}_{ij} = m_{ij}]$ is an indicator function that returns one if $\hat{m}_{ij} = m_{ij}$ and zero otherwise. We reward the increase in map accuracy from time $t-1$ to t:

$$R_t^{anticp} = \text{Accuracy}(\hat{m}_t, m) - \text{Accuracy}(\hat{m}_{t-1}, m). \tag{17}$$

This function rewards actions leading to correct global map predictions, irrespective of whether the agent actually *observed* those locations. For example, if the agent correctly anticipates free space behind a table and is rewarded for that, it then learns to avoid spending additional time around tables in the future to observe that space directly. Resources can be instead allocated to visiting more interesting regions that are harder to anticipate. Additionally, this reward provides a better learning signal while training under noisy conditions by accounting for mapping errors arising from noisy pose and map predictions. Thus, our approach encourages more intelligent exploration behavior by injecting our anticipated occupancy idea directly into the agent's sequential decision-making.

Exploration and navigation with occupancy anticipation Having defined the core occupancy anticipation components, we now demonstrate how our model can be used to benefit embodied navigation in 3D environments. We consider both exploration (discussed above) and *PointGoal navigation* [52, 53], a.k.a PointNav, where the agent must navigate efficiently to a target specified by a displacement vector from the agent's starting position.

For both tasks, we adapt the state-of-the-art Active Neural SLAM (ANS) architecture [47] that previously achieved the best exploration results in the literature and was the winner of the 2019 Habitat PointNav challenge. However, our anticipation model is generic and can be easily integrated with most map-based embodied navigation models [54, 48, 55].

The ANS model is a hierarchical, modular policy for exploration that consists of a mapper, a planner, a local policy, and a global policy (shown in Fig. 7). Given RGB images, the mapper estimates the egocentric occupancy and agent pose, and then temporally aggregates the maps into a global top-down map using the pose estimates. At regular time intervals Δ, the global policy picks a location on the global map to explore. A shortest-path planner decides what trajectory to take from the current position to the target and picks an intermediate goal (within 1.25m) to navigate to. The local policy then selects actions that lead to the intermediate goal; it gets another intermediate goal upon reaching the current goal. See [47] for details. Critically, and like other prior work, the

model of [47] is supervised to generate occupancy estimates based solely on the *visible* occupancy obtained from the egocentric views.

We adapt ANS by modifying the mapper and the reward function. For the mapper, we replace the projection unit from ANS with our anticipation model (see Fig. 7). Additionally, we account for incorrect occupancy estimates in two ways: (1) we filter out high entropy predictions and (2) we maintain a moving average estimate of occupancy at each location in the global map. For the reward function, we use the anticipation-based reward from Eqn. 17.

We train the exploration policy with our anticipation model end-to-end, as this allows adapting to the changing distribution of the agent's inputs. Both the local and the global reinforcement learning policies are trained with Proximal Policy Optimization (PPO) [41]. In our model, the reward of the global policy is our anticipation-based reward defined in Eqn. 17. This replaces the traditional area-coverage reward used in ANS and other current models [48, 47, 51], which rewards the increment in the actual area seen, not the correctly registered area in the map. The reward for the local policy is simply based on the reduction in the distance to the local goal: $R_t^{local} = d_{t-1} - d_t$, where d is the Euclidean distance between the current position and the local goal.

3.3.5 Learning Intrinsic Rewards for Policy Gradient Methods.

To improve a lifelong learning agent's ability to explore, we developed methods for learning intrinsic reward to drive this exploration. The intrinsic rewards can then sit atop any base learner using policy gradients, improving its lifetime reward.

The Optimal Reward Problem We first introduce some terminology.

- **Agent**: A learning system interacting with an environment. On each step t the agent selects an action a_t and receives from the environment an observation s_{t+1} and an *extrinsic* reward r_{t+1} defined by a task $\mathcal{T}$. The agent chooses actions based on a policy $\pi_\theta(a_t|s_t)$ parameterised by θ.
- **Episode**: A finite sequence of agent-environment interactions until the end of the episode defined by the task. An episode return is defined as: $G^{\text{ep}} = \sum_{t=0}^{T_{\text{ep}}-1} \gamma^t r_{t+1}$, where γ is a discount factor, and the random variable T_{ep} gives the number of steps until the end of the episode.
- **Lifetime**: A finite sequence of agent-environment interactions until the end of training defined by an agent-designer, which can consist of multiple episodes. The *lifetime return* is $G^{\text{life}} = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, where γ is a discount factor, and T is the number of steps in the lifetime.
- **Intrinsic reward**: A reward function $r_\eta(\tau_{t+1})$ parameterised by η, where τ_t is a lifetime history with (binary) episode terminations d_i.

The Optimal Reward Problem [6], illustrated in Figure 8, aims to learn the parameters of the intrinsic reward such that the resulting rewards achieve a learning dynamic for an RL agent that maximises the lifetime (extrinsic) return on tasks drawn from some distribution. Formally, the objective function is defined as:

$$J(\eta) = \mathbb{E}_{\theta_0 \sim \Theta, \mathcal{T} \sim p(\mathcal{T})} \left[\mathbb{E}_{\tau \sim p_\eta(\tau|\theta_0)} \left[G^{\text{life}} \right] \right], \tag{18}$$

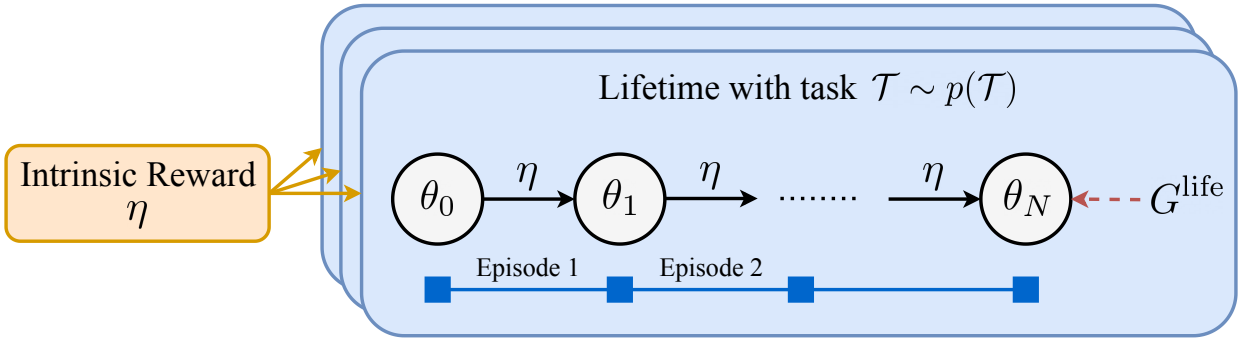


Figure 8: Illustration of the Intrinsic Reward Learning Framework

where Θ and $p(\mathcal{T})$ are an initial policy distribution and a distribution over possibly non-stationary tasks respectively. The likelihood of a lifetime history τ is

$$p_\eta(\tau|\theta_0) = p(s_0) \prod_{t=0}^{T-1} \pi_{\theta_t}(a_t|s_t) p(d_{t+1}, r_{t+1}, s_{t+1}|s_t, a_t)$$

where $\theta_t = f(\theta_{t-1}, \eta)$ is a policy parameter as updated with update function f, which is policy gradient in this paper.[1] Note that the optimisation of η spans multiple lifetimes, each of which can span multiple episodes.

Using the lifetime return G^{life} as the objective instead of the conventional episodic return G^{ep} allows exploration across multiple episodes as long as the lifetime return is maximised in the long run. In particular, when the lifetime is defined as a fixed number of episodes, we find that the lifetime return objective is sometimes more beneficial than the episodic return objective, even for the episodic return performance measure. However, different objectives (e.g., final episode return) can be considered depending on the definition of what a good reward function is.

Meta-Learning Intrinsic Reward We propose a meta-gradient approach [7, 8] to solve the optimal reward problem. At a high-level, we sample a new task $\mathcal{T}$ and a new random policy parameter θ at each lifetime iteration. We then simulate an agent's lifetime by updating the parameter θ using an intrinsic reward function r_η with policy gradient . Concurrently, we compute the meta-gradient by taking into account the effect of the intrinsic rewards on the policy parameters to update the intrinsic reward function with a lifetime value function . Algorithm 4 gives an overview of our algorithm. The following sections describe the details.

Architectures The intrinsic reward function is a recurrent neural network (RNN) parameterised by η, which produces a scalar reward on arriving in state s_t by taking into account the history of an agent's lifetime $\tau_t = (s_0, a_0, r_1, d_1, s_1, ..., r_t, d_t, s_t)$. We claim that giving the lifetime history across episodes as input is crucial for balancing exploration and exploitation, for instance by capturing how frequently a certain state is visited to determine an exploration bonus reward. The lifetime value function is a separate recurrent neural network parameterised by ϕ, which takes the same inputs as the intrinsic reward function and produces a scalar value estimation of the expected future return within the lifetime.

[1] We assume that the policy parameter is updated after each time-step throughout the paper for brevity. However, the parameter can be updated less frequently in practice.

Algorithm 4 Learning intrinsic rewards

Input: $p(\mathcal{T})$: Task distribution
Input: Θ: Randomly-initialised policy distribution
Initialise intrinsic reward η and lifetime value ϕ
repeat
  Initialise task $\mathcal{T} \sim p(\mathcal{T})$ and policy $\theta \sim \Theta$
  while lifetime not ended **do**
    $\theta_0 \leftarrow \theta$
    for $k = 1, 2, \ldots, N$ **do**
      Generate a trajectory using $\pi_{\theta_{k-1}}$
      Update policy $\theta_k \leftarrow \theta_{k-1} + \alpha \nabla_{\theta_{k-1}} J_\eta(\theta_{k-1})$ using intrinsic rewards r_η (Eq. 20)
    Update intrinsic reward function η using Equation 21
    Update lifetime value function ϕ using Equation 23
    $\theta \leftarrow \theta_N$
until η converges

Policy Update Each agent interacts with an environment and a task sampled from a distribution $\mathcal{T} \sim p(\mathcal{T})$. However, instead of directly maximising the extrinsic rewards defined by the task, the agent maximises the intrinsic rewards (r_η) by using policy gradient [56, 57]:

$$J_\eta(\theta) = \mathbb{E}_\theta\left[\sum_{t=0}^{T_{\text{ep}}-1} \bar{\gamma}^t r_\eta(\tau_{t+1})\right] \tag{19}$$

$$\nabla_\theta J_\eta(\theta) = \mathbb{E}_\theta\left[G^{\text{ep}}_{\eta,t} \nabla_\theta \log \pi_\theta(a|s)\right], \tag{20}$$

where $r_\eta(\tau_{t+1})$ is the intrinsic reward at time t, and $G^{\text{ep}}_{\eta,t} = \sum_{k=t}^{T_{\text{ep}}-1} \bar{\gamma}^{k-t} r_\eta(\tau_{k+1})$ is the return of the intrinsic rewards accumulated over an episode with discount factor $\bar{\gamma}$.

Intrinsic Reward and Lifetime Value Update To update the intrinsic reward parameters η, we directly take a meta-gradient ascent step using the overall objective (Eq. 18). Specifically, the gradient is (see the supplementary material for derivation)

$$\nabla_\eta J(\eta) = \mathbb{E}_{\theta_t, \mathcal{T}}\left[G^{\text{life}}_t \nabla_{\theta_t} \log \pi_{\theta_t}(a_t|s_t) \nabla_\eta \theta_t\right], \tag{21}$$

The chain rule is used to get the meta-gradient ($\nabla_\eta \theta_t$) as in previous work [8]. The computation graph of this procedure is illustrated in Figure 8.

Computing the true meta-gradient in Equation 21 requires backpropagation through the entire lifetime, which is infeasible as each lifetime can involve thousands of policy updates. To partially address this issue, we truncate the meta-gradient after N policy updates but approximate the lifetime return $G^{\text{life},\phi}_t \approx G^{\text{life}}_t$ using a *lifetime value function* $V_\phi(\tau)$ parameterised by ϕ, which is learned

using a temporal difference learning from n-step trajectory:

$$G_t^{\text{life},\phi} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(\tau_{t+n}) \tag{22}$$

$$\phi' = \phi + \alpha'(G_t^{\text{life},\phi} - V_\phi(\tau_t))\nabla_\phi V_\phi(\tau_t), \tag{23}$$

where α' is a learning rate. In our empirical work, we found that the lifetime value estimates were crucial to allow the intrinsic reward to perform long-term credit assignments across episodes.

3.3.6 Learning Curriculum Policies for Reinforcement Learning.

The sequence of tasks, or *curriculum*, experienced by a lifelong learner can impact its ability to acquire knowledge and ultimate performance. Here we explore methods for intelligently sequencing these tasks, whenever the task sequence is within the lifelong learner's control.

Curriculum learning is an extension of transfer learning, where the goal is to automatically design and choose a full sequence of tasks (i.e. a *curriculum*) $M_1, M_2, \ldots M_t$ for an agent to train on, such that learning speed or performance on a target task M_t is improved. Transfer learning is leveraged to transfer information between each pair of tasks in this sequence.

Our work builds upon the model proposed by Narvekar and Stone [58], which formulates curriculum generation as an interaction between two agents acting in two different MDPs. One is a *learning agent* that is trying to solve a specific target task MDP M_t, as is the standard case in reinforcement learning. The second is a *curriculum agent*, which interacts in a second, higher level *curriculum MDP*, and whose goal is to sequence tasks M for the learning agent. The way the process unfolds is as follows: the learning agent starts with some initial policy – this is represented as the initial state of the curriculum agent. The curriculum agent selects an action, which corresponds to a task to learn. The learning agent interacts with that task, and updates its policy as a result of learning, which corresponds to a transition in the curriculum agent's state. Learning a task also returns a reward, which is the cost of learning that task. The process terminates once the learning agent learns a policy that can achieve a desired performance threshold on the target task.

This process was defined formally as follows (the superscript C denotes elements of the curriculum MDP; the superscript is dropped when referring to the learning agent trying to solve the task):

Definition 1: A *curriculum MDP (CMDP)* M^C is a 6-tuple $(\mathcal{S}^C, \mathcal{A}^C, p^C, r^C, S_0^C, S_f^C)$, where:

State Space The set of states $\mathcal{S}^C$ consist of the set of all policies π the learning agent can represent, in a form that is executable on the target task. For example, the initial state S_0^C could be the uniform random policy. The terminal states S_f^C are states whose policies achieve a return of at least some desired performance threshold δ on the target task.

Action Space The set of actions $\mathcal{A}^C$, are the prespecified set of tasks a learning agent can train on.

Transition Function The transition function $p^C(s^C, a^C, s'^C)$ gives the probability that s'^C is the learning agent's policy after training on a^C and starting with policy s^C.

Reward Function The reward function $r^C(s^C, a^C)$ is the negative of the time (measured e.g., in experience samples or wall clock time) needed to learn task a^C starting from policy s^C.

A policy $\pi^C : \mathcal{S}^C \mapsto \mathcal{A}^C$ on a CMDP specifies which task to train on given a learning agent policy s^C. Executing π^C for a particular learning agent produces a curriculum. Learning a full policy over a CMDP can be very difficult, due to stochasticity in the learning algorithm (which leads to stochasticity in the CMDP transition function), a very large and continuous state space, and the high cost of taking a CMDP action. Thus, past work on explicit curriculum generation has tried to find traces of specific curricula using approximations and heuristics, rather than learning a full CMDP policy [58, 59]. In this work, we explore the challenges involved in learning π^{C^*}.

Before discussing how to learn a curriculum policy, we first briefly extend the definition of a curriculum MDP. A shortcoming of the previous definition is that it assumes the underlying transfer learning mechanism is value function or policy transfer. Intuitively, the state space of a CMDP represents different states of knowledge. A transition between states reflects the change in knowledge from training on a task and *transferring/incorporating* the information acquired. In value function transfer, the knowledge learned from a task is represented by the value function of the agent itself. However, for transfer via reward shaping, knowledge is represented in terms of a potential-based shaping reward.

Thus, the CMDP state space and transition function are directly related with the transfer learning algorithm being used. The goal of the agent is to reach a state of knowledge that allows solving the target task in the least amount of time.

We now detail how to represent the CMDP state to facilitate learning of curriculum policies. Recall that in the standard reinforcement learning setting, the agent perceives its state as a set of raw state variables. These are typically used to extract basis features $\boldsymbol{\phi}(\mathbf{s})$, which transform the state variables into a space more suitable for learning and for use in function approximation. Given these features and a functional form, the goal is to learn weights $\boldsymbol{\theta}$ for the value function or policy. We introduce an analagous process for curriculum design agents acting in CMDPs. We will ground the discussion assuming the learning agent uses value function transfer. However, the idea is easily applied to the reward shaping setting by noting that the potential-based reward, like the value function, can be expressed as a function of state features and weights.

The first question is how to represent the raw state variables s^C of the CMDP state space. The representation chosen must be able to represent *any* policy the underlying learning agent can represent. Assuming the learning agent derives its policy from an action-value function $Q_{\boldsymbol{\theta}}(s, a)$, the form of the function – in particular, the way values are calculated from $\boldsymbol{\phi}(s, a)$ and $\boldsymbol{\theta}$ (for example, the architecture of a neural network) – determines the class of policies that can be represented. The functional form of $Q_{\boldsymbol{\theta}}(s, a)$ and how learning agent features $\boldsymbol{\phi}$ are extracted are fixed. Thus, it is specific values of the weight vector $\boldsymbol{\theta}$ that actually instantiates a policy in this class. Therefore, it follows that we can represent the state variables for a particular CMDP state s^C using the instantiated vector of learning agent weights $\boldsymbol{\theta}$.

$$s^C = \boldsymbol{\theta} \tag{24}$$

Different instantiations of $\boldsymbol{\theta}$ correspond to different CMDP states. Typically, these weights $\boldsymbol{\theta}$ will take on continuous values. Therefore, in order to learn a CMDP action-value function $Q^C_{\boldsymbol{\theta}^C}(s^C, a^C)$, it will be necessary to do some kind of function approximation. While it is possible to directly use the raw $\boldsymbol{\theta}$ as features for function approximation in the CMDP, learning may be more efficient in an alternative basis space. Thus, it may be beneficial to extract *CMDP basis features* $\boldsymbol{\phi}^C(s^C, a^C)$,

mirroring what is done in the standard MDP setting. For example, with linear value function approximation, $Q^C_{\boldsymbol{\theta}^C}(s^C, a^C) = \boldsymbol{\theta}^C \cdot \boldsymbol{\phi}^C(s^C, a^C)$. The goal then is to learn the weights $\boldsymbol{\theta}^C$ for the CMDP's value function. Any standard RL algorithm can be used to do this.

The questions that remain are: (1) how to convert raw CMDP state variables to CMDP basis features, i.e., the form of $\boldsymbol{\phi}^C(s^C, a^C)$; and (2) what kind of functional form to use to represent the function approximation. The best way to do these will vary by domain. However, in the next 2 subsections, we provide specific examples and guidelines for representations and function approximations that can apply across a broad class of domains.

Discrete State Representations. First we propose one specific way of extracting CMDP state features and performing function approximation, that can be applied when the parameters $\boldsymbol{\theta}$ are tied to specific states, as is common in tabular reinforcement learning. Assume again the learning agent learns an action-value function $Q_{\boldsymbol{\theta}}(s, a)$, for each state-action pair in the task. We can represent Q as a linear function of "one-hot" features $\boldsymbol{\phi}(s, a)$ and their associated weights $\boldsymbol{\theta}$:

$$Q_{\boldsymbol{\theta}}(s, a) = \boldsymbol{\theta} \cdot \boldsymbol{\phi}(s, a) \tag{25}$$

In other words, all the action-values are stored in $\boldsymbol{\theta}$, and $\boldsymbol{\phi}(s, a)$ is a one-hot vector used to select the activated action-value from $\boldsymbol{\theta}$. Our approach for designing $\boldsymbol{\phi}^C$ is to utilize tile coding over subsets of action-values in $\boldsymbol{\theta}$. Specifically, the idea is to create a separate tiling for each primitive state s in the domain. Each such tiling will be defined over the action-values in $\boldsymbol{\theta}$ associated with state s. Thus, this creates $|\mathcal{S}|$ tiling groups, where each group is defined over $|\mathcal{A}|$ CMDP state variables (i.e., action-values). To create the feature space, multiple overlapping tilings are laid over each group.

Since action-values can take a large range of values, we suggest normalizing the action-values within each tiling. Thus, each tiling is over the relative preferences of the different actions in a state. The entire CMDP basis state is the concatenation of all of these tiled features. The effect of this approach is that when computing the value of a CMDP state s^C, the policy for each primitive state contributes equally towards the total value. Two CMDP states will be "closer" in representation space the more $\boldsymbol{\phi}^C$ activates the same tiles – which will happen if they have similar action preferences for primitive states in their task state spaces.

Continuous State Representations. The representation problem is harder in the continuous case, since each parameter θ_i is not local to a state, and we cannot use a state-by-state approach to create a basis feature space. In principle, any continuous feature extraction and function approximation scheme can form the basis of $\boldsymbol{\phi}^C$ (tile coding, neural nets, etc.). We offer 2 guidelines that we found useful in defining successful $\boldsymbol{\phi}^C$ representations in our experiments.

The first is that the precise form of $\boldsymbol{\phi}^C$ should be informed by the domain and the structure of the learning agent's function approximation. The discrete case discussed previously is a special case of this setting. In the discrete case, aggregating action-values in a state-by-state basis could be thought of as exploiting the structure and what we know about the parameter vector $\boldsymbol{\theta}$: namely, that it consists of action values that share states. Depending on the function approximation used by the learning agent, it may be possible to draw similar insights to design $\boldsymbol{\phi}^C$.

The second guideline for creating $\boldsymbol{\phi}^C$ is to capture the *relative* effect of each θ_i on different action preferences. In the discrete case, this was done by normalizing the action values within each

state to create preferences. However, since in general parameters may not be local to a state, the normalization needs to be done directly on the parameter values. In other words, we need to think about how each parameter θ_i affects the policy as a whole over all states, and how each parameter θ_i relates to another. If the parameters $\boldsymbol{\theta}$ are not related, one option would be to create a separate tiling over each parameter, and normalize over all the parameter values. We will demonstrate a specific example of creating $\boldsymbol{\phi}^C$ for the continuous case in the experiments.

Curriculum generalization. In standard reinforcement learning, the value function $v_\pi(s)$ estimates the return of a policy from a given state s. In deep reinforcement learning, the value function is represented by a deep neural network, and exploits the structure in the state space to learn values for observed states and generalize to unseen states. In goal-oriented tasks, where the environment transition dynamics stay the same but the goal state may differ, much of the structure of a value function can also be shared across goals. Thus, the idea behind Universal Value Functions [60] is to create a value function $v_\pi(s, g)$ that generalize over both states s and goals g by creating an embedding over state features and goal features:

$$v_\pi(s, g) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} r_g(s_t, a, s_{t+1}) \middle| s_0 = s \right] \tag{26}$$

Similar ideas can be leveraged to generalize a curriculum policy by learning a universal value function over all the CMDP states and goals. Therefore, we need a way of representing CMDP goals.

A goal in the universal value framework is represented as a single state: $g \in \mathcal{S}$. In the CMDP setting, we instead represent goals g^C as target tasks that the agent could be trained on, with one goal for each target task. Note that this effectively represents a goal as a subset of CMDP states $g^C \subseteq \mathcal{S}^C$, where the student agent policies θ represented by those states are able to solve that specific target task.

A key question is how to represent tasks. To study the generalization of curriculum policy, we restrict our attention to goal-based navigational tasks, which are defined by a starting position and an ending position. This allows us to easily create a parameterized representation of the task by using the concatenated vector of coordinates corresponding to the starting and ending states. However, an important direction for future work is to extend these ideas to non-goal based tasks, such as those described by language or vision commands [61].

Given a representation for both the CMDP state and goal, we use a two-stream neural network architecture as used by [60] to learn a universal value function over the CMDP. A two-stream architecture assumes the problem can be factorized into two components. In our case, one component is $\phi : \mathcal{S}^C \mapsto \mathbb{R}^n$, which creates an embedding for CMDP states. The second is $\psi : \mathcal{G}^C \mapsto \mathbb{R}^n$, which creates an embedding for CMDP goals. The two streams are combined using an output function $h : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^m$. In our case, the mappings ϕ and ψ are represented by multi-layer perceptrons, and the output function is the Hadamard product. See Figure 9 for the architecture we use in the experiments.

A policy extracted from this value function is then able to suggest a task to the student based both on what the student knows, and the task it needs to learn. Given enough experience on a set of "training" target tasks, our experiments will show that learning such a universal value function allows the curriculum policy to generalize and zero-shot produce curricula for unseen "test" target tasks.

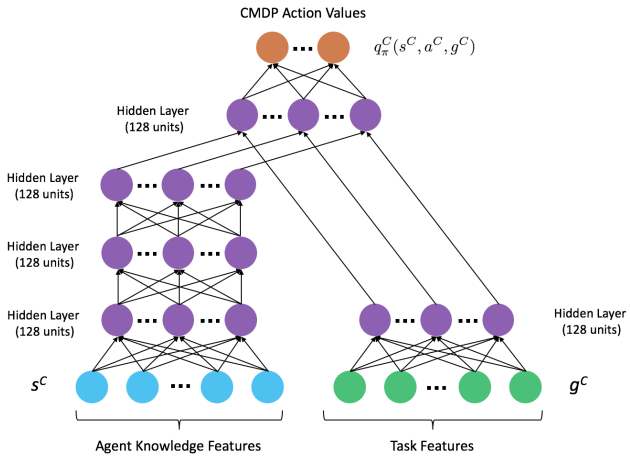


Figure 9: Two-stream Network Architecture used for the Teacher CMDP Agent

3.3.7 Object relationships & distribution models (OBJMAP).

Our goal in this section is to model object relationships to help the agent solve visual scavenger hunt tasks more efficiently. During the agent's exploration of the environment, we want to leverage knowledge about objects that have already been seen to help find target objects more quickly. For example, suppose the target object is a fork, and the robot is in a corridor leading to two doors. Through one door, the robot can see a refrigerator, and through the other door the robot can see a bed. Having learned beforehand relationships among objects will prevent the robot from searching the bedroom unnecessarily.

We address this problem in three steps. First, we learn co-occurrence statistics between different categories of objects. Second, we build a map of the environment containing the location of all the objects we have seen. And finally, we use the two first steps to compute, at each location on the map, the probability for a target object to be present.

Overview of the Method First, to learn the co-occurrence statistics between the different categories of objects, we collect training data in the form of a Boolean table T with dimensions $N \times D$ where N is the number of observations and D the number of object's types. $T[i, j]$ is $True$ if an object of type j appears in observation i, $T[i, j]$ is $False$ otherwise. Details about the data collection method are given in the section *Collecting training data*.

The second component of our method is a map M_o of objects that have already been seen. M_o is an array of dimension $H \times W \times D_o$ where H is the height of the map, W the width of the map and D_o is the number of observable object categories. $D_o[i][j][k]$ contains $True$ if an object of type k is at location i, j, $False$ otherwise. We used the same settings and spatial resolution for the object map and the distribution modeling map as for the occupancy map [29]. Details about the object maps are given in the section *Generating object map*.

Our method outputs a map M_p of dimension $H \times W \times D_u$ where D_u is the the number of unobservable object categories. $M_p[i][j][k]$ contains the probability that an object of type k is at location i, j. Details about the object maps are given in the section *Generating distribution map*.

Experiments setup For the need of the system evaluation we separate objects in two categories: D_o observable objects and D_u unobservable objects. Observable objects are objects that can be detected by the object detection model and added on the objects map. On the other hand, the unobservable objects are objects that can only be found via distribution modelling. Using this formulation, we generated ahead of time the maps with all visible objects for the test environments. (See section *Generate object maps* for more details) In the experiments we use 3 unobservable object categories: Cushion, Cabinet and Picture. All the other object categories are observable.

Collecting training data We collected training data from 88 Habitat-Matterport environments, excluding the 2 test environments that are used for the system evaluation. We randomly sampled 200 locations in each environment. For every location, we took 20 pictures to cover a view at 360 degrees, and we used the object detector described below to collect the list of all the objects visible from this location. For each location (also referred as observation), we added a new row in table T. $T[i][j]$ is $True$ if the object category C_j is visible from the ith location, $T[i][j]$ is $False$ otherwise. In total, we collected 1 760 observations.

Objects detection We used YOLOv5 [62] to detect the objects. We generated the annotations for the training using the semantic segmentation masks provided by Habitat Sim. We converted each semantic region into bounding boxes. For each semantic region, we took the bounding boxes of smallest area that contains the whole semantic region. The semantic labels we trained on are chair, table, sofa, bed, chest of drawers, plant, toilet, stool, towel, tv monitor, shower, bathtub, counter, fireplace, gym equipment, seating, clothes, cushion, cabinet, picture and sink. We trained the object detection model using 88 Habitat-Matterport environments, excluding the 2 test environments. Some results of the object detection model on the testing set are depicted in Figure 10.

Figure 10: Example of Objects Detection Results on Test Environments

Generating object map In this section, we explain how we generated the object maps (maps of observed objects) for the test environments. To generate the object map O_{e_k}, where e_k refers to the kth environment, we randomly sampled 200 locations in e_k. For every location, we took 20 pictures to cover a view at 360 degrees. We used the object detector described in the *Objects detection* section to get the bounding boxes for every detected objects. Using the predicted bounding box, the robot's depth sensor, and the position of the robot in the environment, we converted the coordinates of the middle of the bounding boxes from the image plan to the map coordinates system. We finally

added the object to the map. The object maps for the two test environments can be seen in Figure 11.

Figure 11: Object Maps for Two Test Environments

Generating distribution map For each location (x, y) on the map, we computed the probability of an unobserved object to be present by looking at the observed objects present in a radius of 12px around that location. Let $X_{x,y}^{D_u}$ be the event that object D_u is at location (x, y), and let $L_{x,y}$ be the list of observed object categories within a 12 px radius of (x, y). We can express $P(X_{x,y}^{D_u}|L_{x,y})$ as:

$$P(X_{x,y}^{D_u}|L_{x,y}) = \frac{P(X_{x,y}^{D_u}, L_{x,y})}{P(L_{x,y})} \tag{27}$$

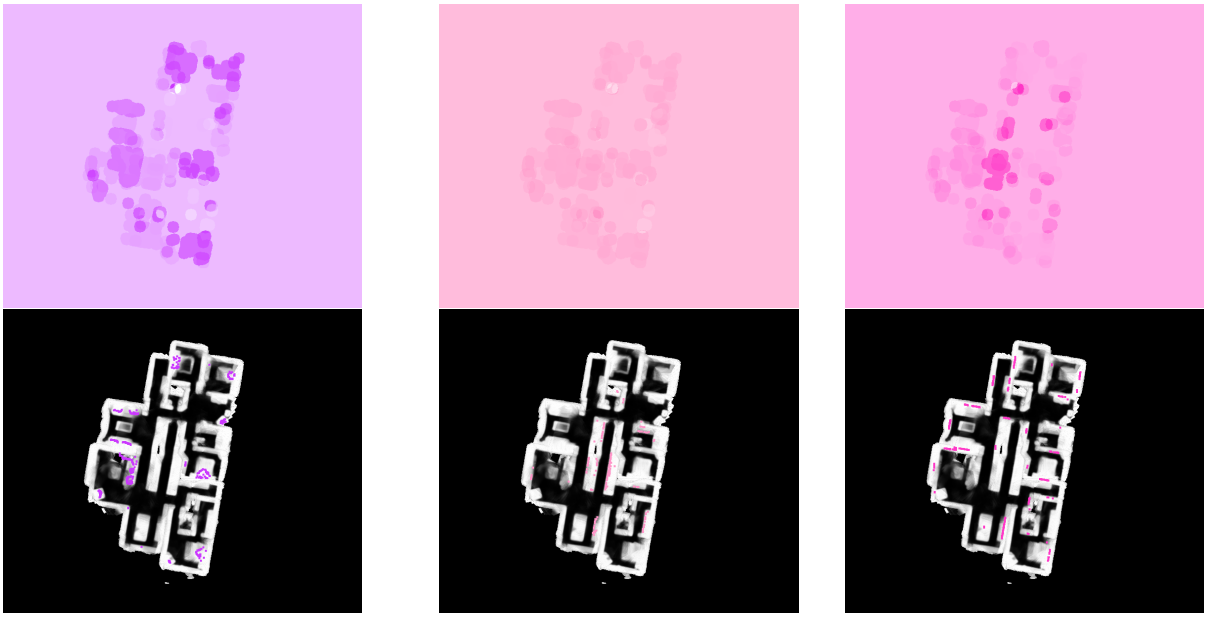
$P(L_{x,y})$ can easily be computed by doing the ratio between $N_{L_{x,y}}$, the number of observations in T that contain all the object categories present in $L_{x,y}$, and N, the total number of observations. More precisely $N_{L_{x,y}}$ can be computed by counting the number of rows in T (see section *Collecting training data* for details about T) where all the columns for objects in $L_{x,y}$ are set to $True$. Similarly $P(X_{x,y}^{D_u}, L_{x,y})$ can be computed by doing the ratio between $N_{D_u,L_{x,y}}$, the observations in T that contain all the object categories present in L as well as D_u, and N. (27) can thus be rewritten as

$$P(X_{x,y}^{D_u}|L_{x,y}) = \frac{N_{L_{x,y}}}{N_{D_u,L_{x,y}}} \tag{28}$$

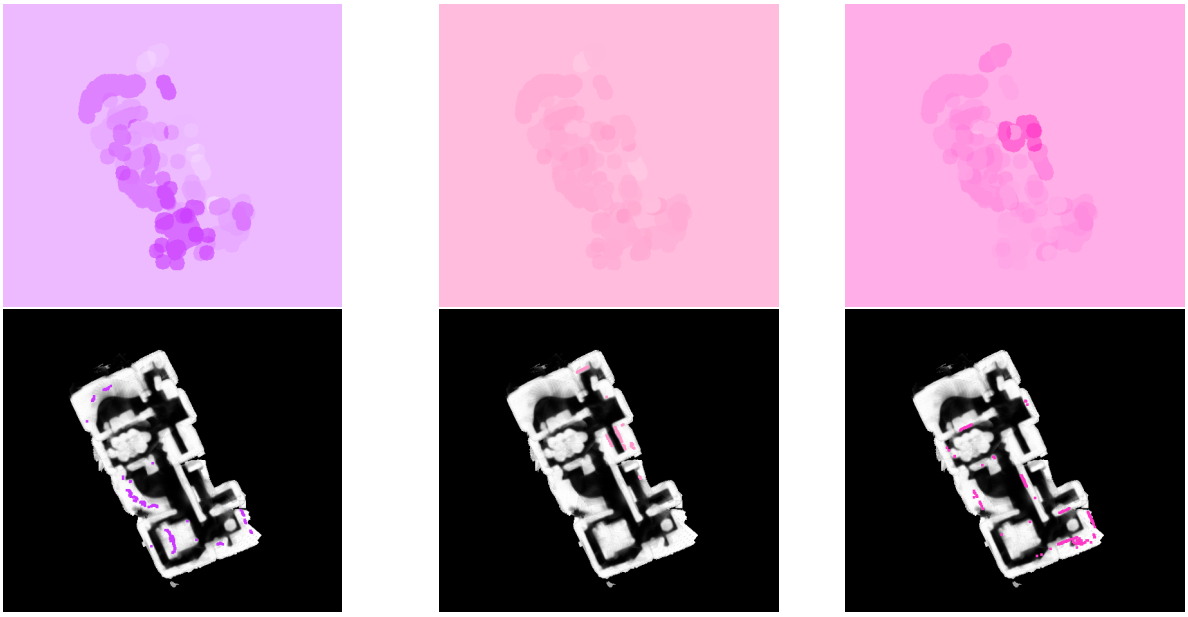
The results for the 3 target objects and the 2 test environments can be seen in Figure 12.

3.4 Details on Other Individual Lifelong Learning Algorithms and Approaches

Our work developed numerous algorithms and methods for different aspects of our lifelong learning approach. Although these methods were not selected for integration into the L2M system for



(a) Cushion (b) Cabinet (c) Picture



(d) Cushion (e) Cabinet (f) Picture

Figure 12: Predicted Distributions (First and Third Rows) and True distributions (Second and Fourth Rows)

the purposes of our evaluation under this project, they are still compatible as components within the system, as described in Table 1 previously. In this section, we detail each of these other problem settings and methods that resulted in major contributions to lifelong learning research. Corresponding evaluation details and results for each approach are provided in Section 4.5. Links to implementations are available in Appendix B.

3.4.1 Compositional Lifelong Classification.

Lifelong compositional learning framework Our framework for lifelong learning of compositional structures (illustrated in Figure 13) stores knowledge in a set of k shared components $M = \{m_1, \ldots, m_k\}$ that are acquired and refined over the agent's lifetime. Each component $m_i = m_{\phi_i} \in \mathcal{M}$ is a self-contained, reusable function parameterized by ϕ_i that can be combined with other components. The agent reconstructs each task's predictive function $f^{(t)}$ via a task-specific structure $s^{(t)} : \mathcal{X}^{(t)} \times \mathcal{M}^k \mapsto \mathcal{F}$, with $\mathcal{M}^k$ being the set of possible sequences of k components, such that $f^{(t)}(\boldsymbol{x}) = s^{(t)}(\boldsymbol{x}, M)(\boldsymbol{x})$, where $s^{(t)}$ is parameterized by a vector $\boldsymbol{\psi}^{(t)}$. Note that $s^{(t)}$ yields a function from $\mathcal{F}$. The structure functions select the components from M and the order in which to compose them to construct the model for each task (the $f^{(t)}$'s).

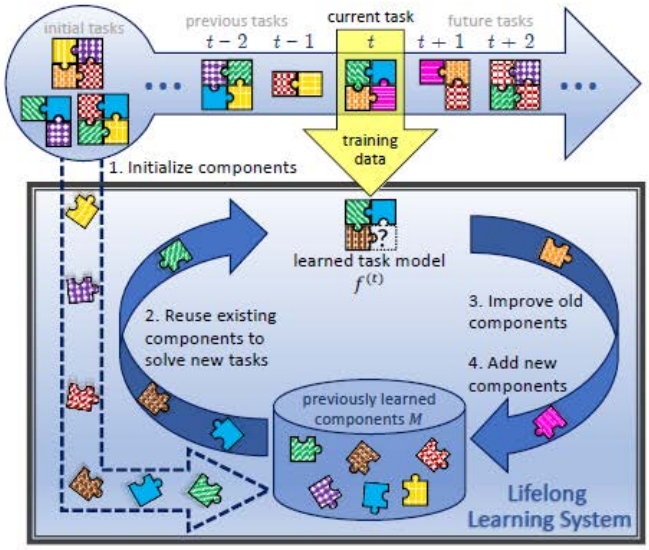


Figure 13: Lifelong Compositional Learning

Algorithm 5 Lifelong Comp. Learning

```
Initialize components M
while Z^(t) ← getTask()
    Freeze M
    for i = 1, ..., structUpdates
        Assimilation step on structure s^(t)
        if i mod adaptFreq = 0
            Freeze s^(t), unfreeze M
            for j = 1, ..., compUpdates
                Adaptation step on M
            Freeze M, unfreeze s^(t)
    Add components via expansion
    Store info. for future adaptation
```

The intuition behind our framework is that, at any point in time t, the agent will have acquired a set of components suitable for solving tasks it encountered previously ($\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(t-1)}$). If these components, with minor adaptations, can be combined to solve the current task $\mathcal{Z}^{(t)}$, then the agent should first learn how to reuse these components before making any modifications to them. The rationale for this idea of keeping components fixed during the early stages of training on the current task $\mathcal{Z}^{(t)}$, before the agent has acquired sufficient knowledge to perform well on $\mathcal{Z}^{(t)}$, is that premature modification could be catastrophically damaging to the set of existing components. Once the structure $s^{(t)}$ has been learned, we consider that the agent has captured sufficient knowledge about the current task, and it would be sensible to update the components to better accommodate that knowledge. If, instead, it is not possible to capture the current task with the existing components, then new components should be added. These notions loosely mirror the stages of assimilation and accommodation in Piaget's ([63]) theories on intellectual development, and so we adopt those terms. Algorithms under our framework take the form of Algorithm 5, split into the following steps.

- **Initialization** The components M should be initialized encouraging reusability, both across tasks and within different structural configurations of task models. The former signifies that

the components should solve a particular sub-problem regardless of the objective of the task. The latter means that components may be reused multiple times within the structure for a single task's model, or at different structural orders across different tasks. For example, in deep nets, this means that the components could be used at different depths. We achieve this by training the first few tasks the agent encounters jointly to initialize M, keeping a fixed, but random, structure that reuses components to encourage reusability.

- **Assimilation** Algorithms for finding compositional knowledge vary in how they optimize each task's structure. In modular nets, component selection can be learned via reinforcement learning [64, 65, 66, 67], stochastic search [68, 69], or backpropagation [70, 71, 72]. Our framework will use any of these approaches to assimilate the current task by keeping the components M fixed and learning only the structure $s^{(t)}$. Approaches supported by our framework must accept decoupling the learning of the structure from the learning of the components themselves; this requirement holds for all the above examples.

- **Accommodation** An effective approach should maintain performance on earlier tasks, while being flexible enough to incorporate new knowledge. To accommodate new knowledge from the current task, the learner may *adapt* existing components or *expand* to include new components:
  - *Adaptation step* Approaches for non-compositional structures have been to naïvely fine-tune models with data from the current task, to impose regularization to selectively freeze weights [73, 74], or to store a portion of data from previous tasks and use experience replay [75, 76]. We will instantiate our framework by using any of these methods to accommodate new knowledge into existing components once the current task has been assimilated. For this to be possible, we require that the method can be selectively applied to only the component parameters $\boldsymbol{\phi}$.
  - *Expansion step* Often, existing components, even with some adaptation, are insufficient to solve the current task. In this case, the learner would incorporate novel components, which should encode knowledge distinct from existing components and combine with those components to solve the new task. The ability to discover new components endows the learner with the flexibility required to learn over a lifetime. For this, we create *component dropout*, described in a subsequent paragraph.

Compositional structures We now present three compositional structures that can be learned within our framework.

- **Linear combinations of models** In the simplest setting, each component is a linear model, and they are composed via linear combinations. Specifically, we assume that $\mathcal{X}^{(t)} \subseteq \mathbb{R}^d$, and each task-specific function is given by $f_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}) = {\boldsymbol{\theta}^{(t)}}^\top \boldsymbol{x}$, with $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$. The predictive functions are constructed from a set of linear component functions $m_{\boldsymbol{\phi}_i}(\boldsymbol{x}) = {\boldsymbol{\phi}_i}^\top \boldsymbol{x}$, with $\boldsymbol{\phi}_i \in \mathbb{R}^d$, by linearly combining them via a task-specific weight vector $\boldsymbol{\psi}^{(t)} \in \mathbb{R}^k$, yielding: $f^{(t)}(\boldsymbol{x}) = s_{\boldsymbol{\psi}^{(t)}}(\boldsymbol{x}, M)(\boldsymbol{x}) = {\boldsymbol{\psi}^{(t)}}^\top(\boldsymbol{\Phi}^\top \boldsymbol{x})$, where we have constructed the matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_k]$ to collect all k components.

- **Soft layer ordering** In order to handle more complex models, we construct compositional deep nets that compute each layer's output as a linear combination of the outputs of multiple modules. As proposed by [72], we assume that each module is one layer, the number of components matches the network's depth, and all components share the input and output dimensions. Concretely, each component is a deep net layer $m_{\boldsymbol{\phi}_i}(\boldsymbol{x}) = \sigma(\boldsymbol{\phi}_i{}^\top \boldsymbol{x})$, where σ is any nonlinear activation and $\boldsymbol{\phi}_i \in \mathbb{R}^{\tilde{d} \times \tilde{d}}$. A set of parameters $\boldsymbol{\psi}^{(t)} \in \mathbb{R}^{k \times k}$ weights the output of the components at each depth: $s^{(t)} = \mathcal{D}^{(t)} \circ \sum_{i=1}^{k} \boldsymbol{\psi}_{i,1}^{(t)} m_i \circ \cdots \circ \sum_{i=1}^{k} \boldsymbol{\psi}_{i,k}^{(t)} m_i \circ \mathcal{E}^{(t)}$, where $\mathcal{E}^{(t)}$ and $\mathcal{D}^{(t)}$ are task-specific input and output transformations such that $\mathcal{E}^{(t)} : \mathcal{X}^{(t)} \mapsto \mathbb{R}^{\tilde{d}}$ and $\mathcal{D}^{(t)} : \mathbb{R}^{\tilde{d}} \mapsto \mathcal{Y}^{(t)}$, and the weights are restricted to sum to one at each depth j: $\sum_{i=1}^{k} \boldsymbol{\psi}_{i,j}^{(t)} = 1$.
- **Soft gating** In the presence of large data, it is often beneficial to modify the network architecture for each input $\boldsymbol{x}$ [65, 71], unlike both approaches above which use a constant structure for each task. We modify the soft layer ordering architecture by weighting each component's output at depth j by an input-dependent soft gating net $s_j^{(t)} : \mathcal{X}^{(t)} \mapsto \mathbb{R}^k$, giving a predictive function $s^{(t)} = \mathcal{D}^{(t)} \circ \sum_{i=1}^{k} [s_1^{(t)}(\boldsymbol{x})]_i m_i \circ \cdots \circ \sum_{i=1}^{k} [s_k^{(t)}(\boldsymbol{x})]_i m_i \circ \mathcal{E}^{(t)}$. As above, we restrict the weights to sum to one at each depth: $\sum_{i=1}^{k} [s_j^{(t)}(\boldsymbol{x})]_i = 1$.

Expansion of the set of components M via component dropout To enable our deep learners to discover new components, we created an expansion step where the agent considers adding a single new component per task. In order to assess the benefit of the new component, the agent learns two different networks: with and without the novel component. Dropout enables training multiple neural networks without additional storage [77], and has been used to prune neural net nodes in non-compositional settings [78]. Our proposed dropout strategy deterministically alternates backpropagation steps with and without the new component, which we call *component dropout*. Intermittently bypassing the new component ensures that existing components can compensate for it if it is discarded. After training, we apply a *post hoc* criterion (in our experiments, a validation error check) to potentially prune the new component.

3.4.2 Unsupervised Hard Example Mining from Videos for Improved Object Detection (DETFLICK).

This section discusses methods for automatically mining hard examples from videos, including data collection, our hard negative mining algorithm, statistics of recovered hard negatives and extension to hard positives.

Video Collection To mine hard examples for face detection, we used 101 videos from sitcoms, each with a duration of 21-25 minutes and a full-length movie of 1 hour 47 minutes, *"Hannah and her sisters"* [79]. Further, we performed YouTube searches with keywords based on: *public address*, *debate society*, *orchestra performance*, *choir practice* and *courtroom*, downloading 89 videos of durations ranging from 10 to 25 minutes. We obtained videos that were expected to feature a large number of human faces in various scenes, reflecting the everyday settings of our face benchmarks. Similarly, for pedestrian detection, we collected videos from YouTube by searching with the two key phrases: *driving cam videos* and *walking videos*. We obtained 40 videos with an average duration of about 30 minutes.

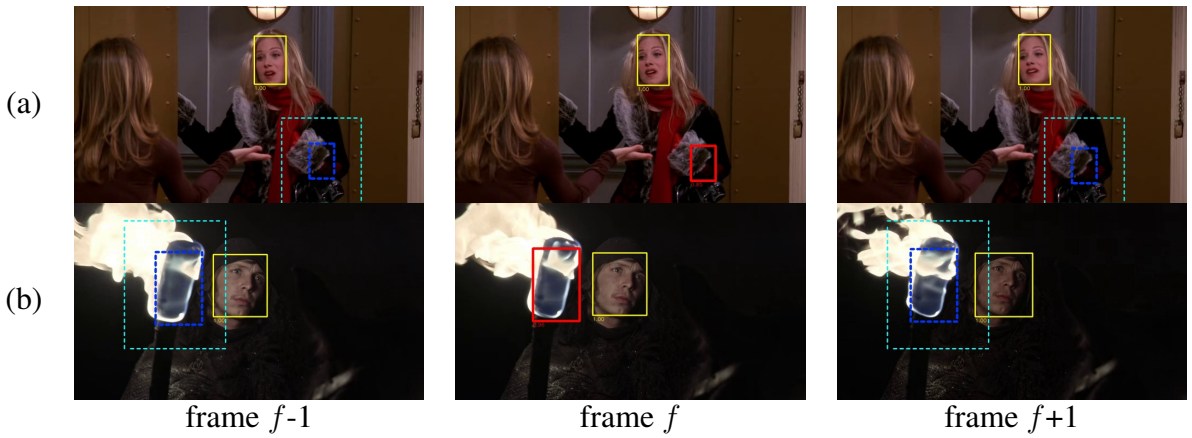


Figure 14: Mining Hard Negatives from Detector-Flicker

Hard Negative Mining Running a pre-trained face detector on every frame of a video gives us a large set of detections with noisy labels. We crucially differ here from recent bootstrapping approaches [80, 81] by (a) using large amounts of *unlabeled* data available on the web instead of relying only on the limited fully-supervised training data from WIDER Face [3] or Caltech Pedestrians [2], and (b) having a novel filtering criterion on the noisy labels obtained from the detector that retains the hard negative examples and minimizes noise in the obtained labels.

The raw detections from a video were thresholded at a relatively high confidence score of 0.8. For every detection in a frame, we formed a short tracklet by performing template matching in adjacent frames, within a window of ± 5 frames — the bounding box of the current detection was enlarged by 100 pixels and this region was searched in adjacent frames for the best match using normalized cross correlation (NCC). To account for occlusions, we put a threshold on the NCC similarity score (set as 0.5) to reject cases where there was a lot of appearance-change between frames. Now in each frame, if the maximum intersection-over-union (IoU) between the tracklet prediction and detections in the adjacent frames was below 0.2, we considered it to be an isolated detection resulting from **detector flicker**. These isolated detections were taken as ***hard negatives***. The detections that *were* found to be consistent with adjacent frames were considered to have a high probability of being true predictions and were termed ***pseudo-positives***. For the purpose of creating the re-training set, we kept only those frames that had at least one pseudo-positive detection in addition to one or more hard negatives. Illustrative examples of this procedure are shown in Figure 14, where we visualize only the previous and next frames for simplicity. The solid boxes denote detections, and the dashed boxes are associated with the tracking algorithm. Given all of the high-confidence **face detections** in a video (**yellow** boxes), the proposed algorithm generates a **tracklet** (**blue** *dashed* boxes) for the **current detection** (**red** box in frame f) by applying template matching within the **search regions** of the adjacent frames (**cyan** *dashed* boxes). As there are no matching detections in adjacent frames for the current detection (i.e., no yellow box matches the blue dashed boxes in frames f-1 or f+1), it is correctly considered to be an "isolated detection" and added to the set of ***hard negatives***. The remaining detections in frame f, which are temporally consistent, are added to the set of ***pseudo-positives***.

Results of Automatic Hard Negative Mining Our initial mining experiments were performed using a standard Faster R-CNN detector trained on WIDER Face [3] for faces and Caltech [2] for

frame f-2 frame f-1 frame f frame f+1 frame f+2

Figure 15: Hard Positive Samples

pedestrians. We collected 13,888 video frames for faces, where each frame contains at least one pseudo-positive and one hard negative (detector flicker). To verify the quality of our automatically mined hard negatives, we randomly sampled 511 hard negatives for inspection. 453 of them are true negatives, while 16 samples are true positives, and 42 samples are categorized as *ambiguous*, which correspond extreme head pose or severe occlusions. The precision for true negatives is 88.65% and precision for true negatives plus *ambiguous* is 96.87%.
For pedestrians, we collected 14,967 video frames. We manually checked 328 automatically mined hard negatives, where 244 of them are true negatives and 21 belong to *ambiguous*. The precision for true negatives is 74.48% and precision for true negatives plus *ambiguous* is 82.18%.
To further validate our method on an existing fully-annotated video dataset, we used the Hannah dataset [79], which has every frame annotated with face bounding boxes. Here, out of 234 mined hard negatives, 187 were true negatives, resulting in a precision of 79.91%. We note that the annotations on the Hannah movie are not always consistent and involve a significant domain shift from WIDER. Considering the fact no human supervision is provided, the mined face hard negatives are consistently of high quality across various domains.

Extension to Hard Positive Mining In principle, the same concept for using detector flickers can be directly applied to obtaining ***hard positives***. The idea is to look for "off-flickers" of a detector in a video tracklet – given a series of detections of an object in a video, such as a face, we can search for single frames that have no detections but are surrounded by detections on either side. Of course, these could be caused by short-duration occlusions, for example, but a large percentages of these "off-flickers" are hard positives, as in Figure 15. We generate tracklets using the method from [82] and show results incorporating hard positives on pedestrian and face detection in the experiments section. The manually calculated purity over 300 randomly sampled frames was 94.46% for faces and 83.13% for pedestrians.

3.4.3 Automatic adaptation of object detectors to new domains using self-training (STSL).

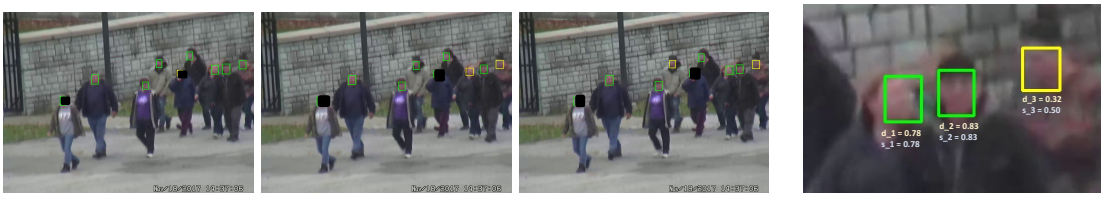
In the next paragraphs, we will describe the procedures to automatically label the target domain, to re-train using these pseudo-labels, and to create soft-labels.

Automatic Labeling of the Target Domain Self-labeling [83] or pseudo-labeling [84] adapts a pre-existing or *baseline* model, trained on a labeled *source* domain $\mathcal{S}$, to a novel unlabeled *target* domain $\mathcal{T}$, by treating the model's own predictions on the new dataset as training labels. In our case, we obtain target domain pseudo-labels by selecting high-confidence predictions of the baseline detector, followed by a refinement step using a tracker.

Pseudo-labels from detections. The baseline detector is run on every frame of the unlabeled videos in the target domain and if the (normalized) detector confidence score for the i-th prediction (i.e.,the model's posterior), d_i, is higher than some threshold θ, then this prediction is added to the set of pseudo-labels.

In practice, we select 0.5 for θ for face detection and 0.8 for person detection. Note that such a threshold is easily selected by visually inspecting a small number of unlabeled videos from $\mathcal{T}$ (5 videos); we compare with a fully-automated procedure in the results section.

Refined labels from tracking. Exploiting the temporal continuity between frames in a video, we can enlarge our set of pseudo-labels with objects missed by the baseline detector. To link multiple object detections across video frames into temporally consistent tracklets, we use the algorithm from Jin et al. (Sec. 3 of [82]) with the MD-Net tracker [85]. Now, given a tracklet that consistently follows an object through a video sequence, when the object detector did not fire (i.e.,$d_i < \theta$) in some difficult frames, the tracker can still correctly predict an object (see Fig. 16(a)[2]). We expand the set of pseudo-labels to include these "tracker-only" bounding-boxes that were missed by the baseline detector, since these *hard examples* are expected to have a larger influence on the model's decision boundary upon retraining [86, 87, 11]. Further, we prune out extremely short tracklets (less than 10 frames) to remove the effects caused by spurious detections.



(a) (b)

Figure 16: (a) Pseudo-labels from Detection and Tracking (b) Soft-Labeling Example

Training on pseudo-labels We use the popular Faster R-CNN (FRCNN) [88, 89] as our detector. In a naive setting, we would treat both labeled source-domain data and pseudo-labeled target-domain data identically in terms of the loss. We give a label of 1 to *all* the target domain pseudo-labeled samples, irrespective of whether it originated from the baseline detector or the tracker – i.e.,for X_i, the i-th training sample drawn from $\mathcal{T}$, the label y_i is defined as

$$y_i = \begin{cases} 1, & \text{if } X_i \text{ is a } \textit{pos.} \text{ sample (from detector or tracker).} \\ 0, & \text{if } X_i \text{ is a } \textit{neg.} \text{ sample.} \end{cases} \tag{29}$$

Note that here X_i is not an image, but a *region* in an image. For training the classification branch, we use a binary cross-entropy loss on the i-th training sample:

$$\mathcal{L}_i(y_i, p_i) = -[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{30}$$

where "*hard*" label $y_i \in \{0, 1\}$ and the model's predicted posterior $p_i \in [0, 1]$. This is similar to the method of Jin et al. [11], which assigns a label of 1 for both easy and hard positive examples during re-training.

[2]Some faces hidden following permissions in [1].

Distillation loss with soft labels For training data coming from $\mathcal{T}$, many of the y_is can be noisy, so a "*soft*" version of the earlier $\{0, 1\}$ labels could help mitigate the risk from mislabeled target data. Label smoothing in this fashion has been shown to be useful in generalization [90, 91], in reducing the negative impact of incorrect training labels [92] and is more informative about the distribution of labels than one-hot encodings [93]. In our case, each target-domain *positive* label can have two possible origins – (i) high-confidence predictions from the baseline detector or (ii) the tracklet-formation process. We assign a ***soft score*** s_i to each positive target-domain sample $X_i \in \mathcal{T}$ as follows:

$$s_i = \begin{cases} d_i, & \text{if } X_i \text{ originates from detector.} \\ \theta, & \text{if } X_i \text{ originates from tracker.} \end{cases} \tag{31}$$

For a pseudo-label originating from the baseline detector, a high detector confidence score d_i is a reasonable measure of reliability. Tracker-only pseudo-labels, which could be objects missed by the baseline model, are emphasized during training – their soft score is raised up to the threshold θ, although the baseline's confidence on them had fallen below this threshold. An illustrative example is shown in Figure 16(b).

Label interpolation. A ***soft label*** $\tilde{y}_i$ is formed by a linear interpolation between the earlier hard labels y_i and soft scores s_i, with $\lambda \in [0, 1]$ as a tunable hyper-parameter.

$$\tilde{y}_i = \lambda s_i + (1 - \lambda) y_i \tag{32}$$

The loss for the i-th positive sample now looks like

$$\mathcal{L}_i^{distill} = \begin{cases} \mathcal{L}_i(y_i, p_i), & \text{if } X_i \in \mathcal{S}. \\ \mathcal{L}_i(\tilde{y}_i, p_i), & \text{if } X_i \in \mathcal{T}. \end{cases} \tag{33}$$

Setting a high value of λ creates softer labels $\tilde{y}_i$, trusting the baseline source model's prediction s_i more than than the riskier target pseudo-labels y_i. In this conservative setting, the softer labels will decrease the overall training signal from target data, but also reduces the chance of incorrect pseudo-labels having a large detrimental effect on the model parameters.

We now describe two schemes to avoid explicitly depending on the λ hyper-parameter –

I. Constrained hard examples. Assigning a label of 1 to both "easy" and "hard" examples (i.e.,high-confidence detections and tracker-only samples), as in Sec. 3.4.3, gives equal importance to both. Training with *just* the hard examples can be sub-optimal – it might decrease the model's posteriors on instances it was getting correct initially. Ideally, we would like to emphasize the hard examples, while simultaneously *constraining* the model to maintain its posteriors on the other (easy) samples. We can achieve this by setting $\theta = 1$ in Equation 31 and $\lambda = 1$ in Equation 32, which would create a label of 1 for tracker-only "hard" examples, and a label equal to baseline detector score for the high-confidence detections, i.e.,"easy" examples.

II. Cross-domain score mapping. Let us hypothetically consider what the distribution of detection scores on $\mathcal{T}$ would be like, had the model been trained on *labeled* target domain data. With minimal information on $\mathcal{T}$, it is reasonable to assume this distribution of scores to be similar to that on $\mathcal{S}$. The latter is an "ideal" operating condition of training on labeled data and running inference on within-domain images. Let the actual distribution of baseline detector scores on $\mathcal{T}$ have p.d.f. $f(x)$, and the

distribution of scores on $\mathcal{S}$ have p.d.f. $g(x)$. Let their cumulative distributions be $F(x) = \int_0^x f(t)dt$ and $G(x) = \int_0^x g(r)dr$, respectively. As a parameter-free method of creating soft-labels for our pseudo-labels on $\mathcal{T}$, we can use histogram specification [94] to map the baseline detector scores on $\mathcal{T}$ to match the distribution of scores on images from $\mathcal{S}$, i.e.,replace each target domain score x with $G^{-1}(F(x))$. The inverse mapping is done through linear interpolation. Figure 17(a) shows the distribution of scores for a model trained on labeled WIDER-Face [3] and run on images from the validation split of the same dataset. In Figure. 17(b), due to the domain shift, there is a visible difference when this model is run on unlabeled images from CS6 surveillance videos [1]. Figure 17(c) shows the effect of histogram matching. Concretely, detector samples get soft-label $G^{-1}(F(d_i))$, while tracker-only samples get soft-label θ.

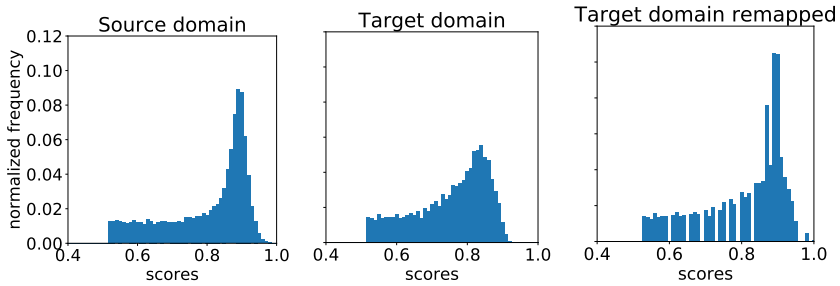


Figure 17: Cross-domain Score Mapping (a) WIDER-Validation (b) CS6 Surveillance Videos [1] (c) Remapping the Scores on CS6 to Resemble WIDER

3.4.4 Half&Half: New Tasks and Benchmarks for Studying Visual Common Sense (HNH).

We define three different Half&Half tasks (Fig. 18):

- **Image-to-Label task:** One half of the image is provided, and the task is to infer a categorical label for what is likely to be present in the other half amongst the given K choices(Fig. 19).
- **Label-to-Image task:** A target category and a set of K half images are provided. The task is to infer which candidate is most likely to have the target in its other half (Fig. 20).
- **Image-to-Image task:** A query image and K image choices are provided, all of them being half images. The task is to infer which of the choices is the most likely to be from the same image as the query (Fig. 21).

The three variants of the tasks were inspired by the common-sense reasoning capabilities of intelligent beings under uncertainty. Specifically, an agent trying to find a specific type of object should be able to decide whether the current direction is promising (Image-to-Label). And if not, given observations towards other directions, which one should be preferred (Label-to-Image)? Image-to-Image is modeling an intelligent reasoning capability to directly predict the next visual observations, which can enable an agent to prepare for imminent encounters.

Our hope is that the Half&Half benchmarks, and perhaps their next generations, drive forward the research in designing intelligent agents by training and evaluating such systems for visual "common sense". We aim to make this benchmark public soon and plan to keep a public leaderboard for the benchmarks.

Figure 18: The Half&Half Visual Prediction Tasks

The Half&Half Benchmarks In this section, we describe our three new benchmarks. Each benchmark is constructed to study one of the three variants of the Half&Half tasks introduced previously. We make use of images and annotations from existing datasets originally created for object detection or scene understanding. As we will show in this section, the way we create the benchmarks requires no additional annotations compared to those standard recognition tasks. This allows us to directly make use of large-scale existing datasets.

Figure 19: Half&Half Image-to-Label Benchmark Example

The Image-to-Label benchmark

Image selection The benchmark is created using the training and validation images from MS-COCO [95]. We consider the left half of each image as the context for the objects present in the

right half . From the above sets, we sample images that have at least one single object present in its right half. Furthermore, we discard images whose left half and right half contain any overlapping objects. As a design choice, we exclude the "person" category and consider only the remaining 79 categories since we observe that "person" is very common in MS-COCO and has significant co-occurrence with the majority of other categories. In total, we obtain $45,843$ images meeting the criteria above.

Problems and splits Out of all the obtained images, we create a random train/val/test split of $32,000$/$3,843$/$10,000$ images. Each of the training and validation images are provided with one image (the left half) and a set of labels from the right half. From the test images, we form test problems in the form of the Image-to-Label task. Figure 19 shows an example. Five candidate categories are given where only one of them actually appears in the right half. For the correct candidate choice, we randomly pick one object category that exists in the right half among the ground truth. For the wrong candidates, we randomly select from all MS-COCO object categories not present in the whole image.

Evaluation During testing, we evaluate the performance of a model on the test problems based on whether it can pick the right candidate among the five choices, and also the rank that it assigns to the correct candidate. Specifically, benchmark users are required to report:

(1) Rank-1 Accuracy: $\frac{1}{N}\sum_i [\![r_i = 1]\!]$,
(2) Mean Reciprocal Rank (MRR): $\frac{1}{N}\sum_i \frac{1}{r_i}$.

Here N denotes the total number of test samples and r_i is the rank of the correct candidate in a model's output.

The Label-to-Image benchmark

Image selection Because of the close formulation between the Label-to-Image and Image-to-Label tasks, we can reuse the data collected for the Image-to-Label benchmark, with a few critical modifications. The same set of images, labels, and train/val/test image split are used. The differences only lie in the way the problems are formulated.

Problems and splits As illustrated in Figure 20, a Label-to-Image problem contains a query label and 10 gallery images. We create one such problem for each of the images in the benchmark using the following steps:

(1) Each (right-half object label, left-half image) pair from Image-to-Label benchmark is sampled as query object and correct candidate;
(2) From the remaining images in the split, images containing the query object label in their right halves are filtered out. The remaining left-half images are then ranked based on the similarity scores with the correct candidate and 9 images are selected randomly as wrong candidates from the top 100.

We follow [96] and use low-level visual features (GIST and color histogram) for computing the similarity. In total, we obtain $47,370$/$5,686$/$10,000$ train/val/test problem sets.

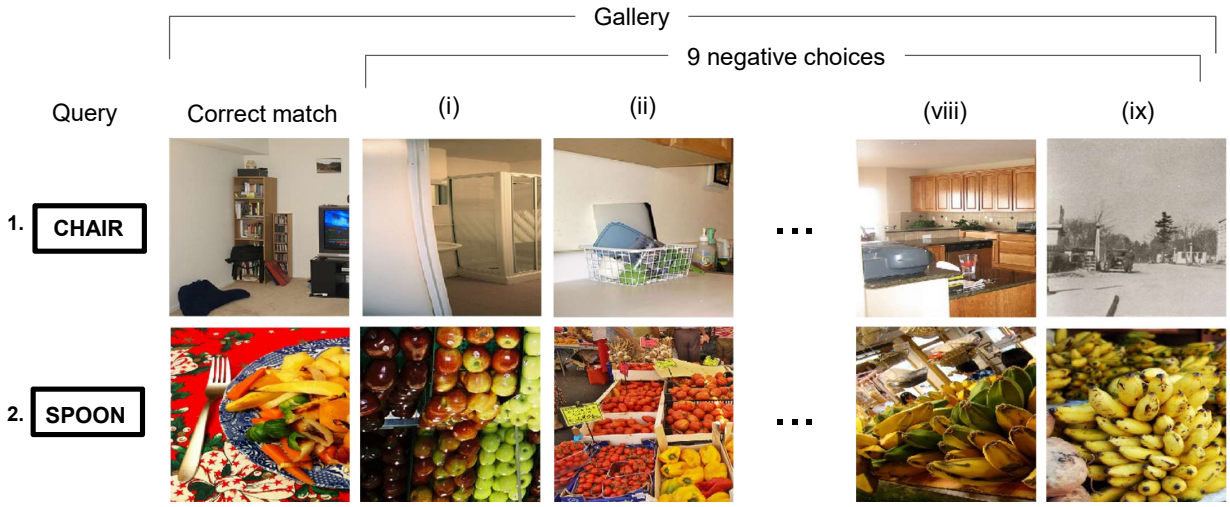


Figure 20: Half&Half Label-to-Image Benchmark Examples

Evaluation During testing, the objective is to correctly pick the correct choice among the $K = 10$ gallery choices. The evaluated algorithm is required to provide a ranking among the choices for each test problem, and the two evaluation measures, rank-1 accuracy and MRR, are reported.

The Image-to-Image benchmark

Image selection Since the Image-to-Image task involves half images as both queries and candidate choices and requires no label annotations, we are able to consider any natural images for building the benchmark. We chose to use the SUN360 dataset [97], which offers a large collection of high-resolution 9104×4552 rectangular panorama images. Using panoramas allows us to cut image crops instead of using adjacent halves. By making the two "halves" some distance apart, they will have little overlap in content and offer diverse visual information.

Figure 21: HNH: Half&Half Image-to-Image Benchmark Example

Problems and splits Among all images available in SUN360, we randomly sample $27,999$ training images and $29,142$ testing images. Each partition is further divided into a query partition (from which query and correct choices are drawn) and a gallery partition (from which negative choices are drawn). We construct one problem for each image in the query partition: one of its two crops is randomly chosen as the query image and the other one as the correct choice. Nine wrong choices are then randomly sampled from the corresponding gallery partition. A ranking procedure that is the same as the Label-to-Image benchmark is also applied to avoid trivial solutions. In total,

we obtain $8,399$ training and $8,742$ testing problems, which are constructed from the training and testing partitions, respectively.

Evaluation During testing, the objective is to correctly pick the correct one among the $K = 10$ choices. The evaluated algorithm is required to pick a top choice for each test problem. Rank-1 accuracy is reported for evaluation.

3.4.5 Efficient Lifelong Inverse Reinforcement Learning.

The inverse RL problem In inverse RL (IRL) [98], the agent does not know the MDP's reward function, and must infer it from demonstrations $\mathcal{Z} = \{\zeta_1, \ldots, \zeta_n\}$ given by an expert user. Each demonstration ζ_j is a sequence of state-action pairs $[\boldsymbol{s}_{0:H}, \mathbf{a}_{0:H}]$ that is assumed to be generated by the user's unknown policy $\hat{\pi}^*$. Once the reward function is learned, the MDP is complete and so can be solved for the optimal policy π^*.

Given an MDP$\backslash \boldsymbol{r} = \langle \mathcal{X}, \mathcal{U}, T, \gamma \rangle$ and expert demonstrations $\mathcal{Z}$, the goal of IRL is to estimate the unknown reward function $\boldsymbol{r}$ of the MDP. Previous work has defined the optimal reward such that the policy enacted by the user be (near-)optimal under the learned reward ($V^{\pi^*} = V^{\hat{\pi}^*}$), while (nearly) all other actions would be suboptimal. This problem is unfortunately ill-posed, since it has numerous solutions, and so it becomes necessary to make additional assumptions in order to find solutions that generalize well. These various assumptions and the strategies to recover the user's policy have been the focus of previous IRL research. We next focus on the MaxEnt approach to the IRL problem.

Maximum Entropy IRL In the maximum entropy (MaxEnt) algorithm for IRL [99], each state s_i is represented by a feature vector $\mathbf{x}_{s_i} \in \mathbb{R}^d$. Each demonstrated trajectory ζ_j gives a *feature count* $\mathbf{x}_{\zeta_j} = \sum_{i=0}^{H} \gamma^i \mathbf{x}_{s_i}$, giving an approximate expected feature count $\tilde{\mathbf{x}} = \frac{1}{n} \sum_j \mathbf{x}_{\zeta_j}$ that must be matched by the agent's policy to satisfy the condition $V^{\pi^*} = V^{\hat{\pi}^*}$. The reward function is represented as a parameterized linear function with weight vector $\boldsymbol{\theta} \in \mathbb{R}^d$ as $\boldsymbol{r}_{s_i} = \boldsymbol{r}(\mathbf{x}_{s_i}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}_{s_i}$ and so the cumulative reward of a trajectory ζ_j is given by $\boldsymbol{r}_{\zeta_j} = \boldsymbol{r}(\mathbf{x}_{\zeta_j}, \boldsymbol{\theta}) = \sum_{s_i \in \zeta_j} \gamma^i \boldsymbol{\theta}^\top \mathbf{x}_{s_i} = \boldsymbol{\theta}^\top \mathbf{x}_{\zeta_j}$.

The algorithm deals with the ambiguity of the IRL problem in a probabilistic way, by assuming that the user acts according to a MaxEnt policy. In this setting, the probability of a trajectory is given as: $P(\zeta_j \mid \boldsymbol{\theta}, T) \approx \frac{1}{Z(\boldsymbol{\theta}, T)} \exp(\boldsymbol{r}_{\zeta_j}) \prod_{(s_i, a_i, s_{i+1}) \in \zeta_j} T(s_{i+1} \mid s_i, a_i)$, where $Z(\boldsymbol{\theta}, T)$ is the partition function, and the approximation comes from assuming that the transition uncertainty has little effect on behavior. This distribution does not prefer any trajectory over another with the same reward, and exponentially prefers trajectories with higher rewards. The IRL problem is then solved by maximizing the likelihood of the observed trajectories $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log P(\mathcal{Z} \mid \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{\zeta_j \in \mathcal{Z}} \log P(\zeta_j \mid \boldsymbol{\theta}, T)$. The gradient of the log-likelihood is the difference between the user's and the agent's feature expectations, which can be expressed in terms of the state visitation frequencies D_s: $\tilde{\mathbf{x}} - \sum_{\tilde{\zeta} \in \mathcal{Z}_{MDP}} P(\tilde{\zeta} \mid \boldsymbol{\theta}, T) \mathbf{x}_{\tilde{\zeta}} = \tilde{\mathbf{x}} - \sum_{s \in \mathcal{X}} D_s \mathbf{x}_s$, where $\mathcal{Z}_{MDP}$ is the set of all possible trajectories. The D_s can be computed efficiently via a forward-backward algorithm [99]. The maximum of this concave objective is then achieved when the feature counts match, and so $V^{\pi^*} = V^{\hat{\pi}^*}$.

The lifelong IRL problem We now introduce the novel problem of lifelong IRL. In contrast to most previous work on IRL, which focuses on single-task learning, this paper focuses on online

multi-task IRL. Formally, in the lifelong learning setting, the agent faces a sequence of IRL tasks $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N_{max})}$, each of which is an MDP$\backslash \boldsymbol{r}$ $\mathcal{T}^{(t)} = \left\langle \mathcal{X}^{(t)}, \mathcal{U}^{(t)}, T^{(t)}, \gamma^{(t)} \right\rangle$. The agent will learn tasks consecutively, receiving multiple expert demonstrations for each task before moving on to the next. We assume that *a priori* the agent does not know the total number of tasks N_{max}, their distribution, or the order of the tasks.

The agent's goal is to learn a set of reward functions $\mathcal{R} = \left\{ \boldsymbol{r}(\boldsymbol{\theta}^{(1)}), \ldots, \boldsymbol{r}(\boldsymbol{\theta}^{(N_{max})}) \right\}$ with a corresponding set of parameters $\Theta = \left\{ \boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(N_{max})} \right\}$. At any time, the agent may be evaluated on any previous task, and so must strive to optimize its performance for all tasks $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}$, where N denotes the number of tasks seen so far ($1 \leq N \leq N_{max}$). Intuitively, when the IRL tasks are related, knowledge transfer between their reward functions has the potential to improve the learned reward function for each task and reduce the number of expert demonstrations needed.

After N tasks, the agent must optimize the likelihood of all observed trajectories over those tasks:

$$\max_{\boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(N)}} P\left(\boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(N)}\right) \prod_{t=1}^{N} \left(\prod_{j=1}^{n_t} P\left(\zeta_j \mid \boldsymbol{r}^{(t)}\right) \right)^{\frac{1}{n_t}} \quad , \tag{34}$$

where $P(\boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(N)})$ is a reward prior to encourage relationships among the reward functions, and each task is given equal importance by weighting it by the number of associated trajectories n_t.

ELIRL As described above, the lifelong IRL agent must optimize its performance over all IRL tasks observed so far. Using the MaxEnt assumption that the reward function $\boldsymbol{r}_{s_i}^{(t)} = \boldsymbol{\theta}^{\top} \mathbf{x}_{s_i}^{(t)}$ for each task is linear and parameterized by $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$, we can factorize these parameters into a linear combination $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$ to facilitate transfer between parametric models, following Kumar and Daumé [100] and Maurer et al. [34]. The matrix $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ represents a set of k latent reward vectors that are shared between all tasks, with sparse task-specific coefficients $\boldsymbol{s}^{(t)} \in \mathbb{R}^k$ to reconstruct $\boldsymbol{\theta}^{(t)}$.

Using this factorized representation to facilitate transfer between tasks, we place a Laplace prior on the $\boldsymbol{s}^{(t)}$'s to encourage them to be sparse, and a Gaussian prior on $\boldsymbol{L}$ to control its complexity, thereby encouraging the reward functions to share structure. This gives rise to the following reward prior:

$$P\left(\boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(N)}\right) = \frac{1}{Z\left(\lambda, \mu\right)} \exp\left(-N\lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2\right) \prod_{t=1}^{N} \exp\left(-\mu \|\boldsymbol{s}^{(t)}\|_1\right) \quad , \tag{35}$$

where $Z(\lambda, \mu)$ is the partition function, which has no effect on the optimization. We can substitute the prior in Equation 35 along with the MaxEnt likelihood into Equation 34. After taking logs and re-arranging terms, this yields the equivalent objective:

$$\min_{\boldsymbol{L}} \frac{1}{N} \sum_{t=1}^{N} \min_{\boldsymbol{s}^{(t)}} \left\{ -\frac{1}{n_t} \sum_{\zeta_j^{(t)} \in \mathcal{Z}^{(t)}} \log P\left(\zeta_j^{(t)} \mid \boldsymbol{L}\boldsymbol{s}^{(t)}, T^{(t)}\right) + \mu \|\boldsymbol{s}^{(t)}\|_1 \right\} + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \quad . \tag{36}$$

Note that Equation 36 is separably, but not jointly, convex in $\boldsymbol{L}$ and the $\boldsymbol{s}^{(t)}$'s; typical multi-task approaches would optimize similar objectives [100, 34] using alternating optimization.

To enable Equation 36 to be solved online when tasks are observed consecutively, we adapt concepts from the lifelong learning literature. Ruvolo and Eaton [35] approximate a multi-task objective with

a similar form to Equation 36 online as a series of efficient online updates. Note, however, that their approach is designed for the supervised setting, using a general-purpose supervised loss function in place of the MaxEnt negative log-likelihood in Equation 36, but with a similar factorization of the learned parametric models. Following their approach but substituting in the IRL loss function, for each new task t, we can take a second-order Taylor expansion around the single-task point estimate of $\boldsymbol{\alpha}^{(t)} = \arg\min_{\boldsymbol{\alpha}} -\sum_{\zeta_j^{(t)} \in \mathcal{Z}^{(t)}} \log P\big(\zeta_j^{(t)} \mid \boldsymbol{\alpha}, T^{(t)}\big)$, and then simplify to reformulate Equation 36 as

$$\min_{\boldsymbol{L}} \frac{1}{N} \sum_{t=1}^{N} \min_{\boldsymbol{s}^{(t)}} \Big\{ \big(\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)}\big)^{\top} \boldsymbol{H}^{(t)} \big(\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)}\big) + \mu \|\boldsymbol{s}^{(t)}\|_1 \Big\} + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \tag{37}$$

where the Hessian $\boldsymbol{H}^{(t)}$ of the MaxEnt negative log-likelihood is given by:

$$\boldsymbol{H}^{(t)} = \frac{1}{n_t} \nabla^2_{\boldsymbol{\theta},\boldsymbol{\theta}} \mathcal{L}\big(\boldsymbol{r}\big(\boldsymbol{L}\boldsymbol{s}^{(t)}\big), \mathcal{Z}^{(t)}\big) = \left(-\sum_{\tilde{\zeta} \in \mathcal{Z}_{MDP}} \mathbf{x}_{\tilde{\zeta}} P(\tilde{\zeta}|\boldsymbol{\theta}) \right) \left(\sum_{\tilde{\zeta} \in \mathcal{Z}_{MDP}} \mathbf{x}_{\tilde{\zeta}}^{\top} P(\tilde{\zeta}|\boldsymbol{\theta}) \right) + \sum_{\tilde{\zeta} \in \mathcal{Z}_{MDP}} \mathbf{x}_{\tilde{\zeta}} \mathbf{x}_{\tilde{\zeta}}^{\top} P(\tilde{\zeta}|\boldsymbol{\theta}) \ . \tag{38}$$

Since $\boldsymbol{H}^{(t)}$ is non-linear in the feature counts, we cannot make use of the state visitation frequencies obtained for the MaxEnt gradient in the lifelong learning setting. This creates the need for obtaining a sample-based approximation. We first solve the MDP for an optimal policy $\pi^{\boldsymbol{\alpha}^{(t)}}$ from the parameterized reward learned by single-task MaxEnt. We compute the feature counts for a fixed number of finite horizon paths by following the stochastic policy $\pi^{\boldsymbol{\alpha}^{(t)}}$. We then obtain the sample covariance of the feature counts of the paths as an approximation of the true covariance in Equation 38.

Given each new consecutive task t, we first estimate $\boldsymbol{\alpha}^{(t)}$ as described above. Then, Equation 37 can be approximated online as a series of efficient update equations [35]:

$$\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \ell\big(\boldsymbol{L}_N, \boldsymbol{s}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{H}^{(t)}\big) \tag{39}$$

$$\boldsymbol{L}_{N+1} \leftarrow \arg\min_{\boldsymbol{L}} \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 + \frac{1}{N} \sum_{t=1}^{N} \ell\big(\boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{H}^{(t)}\big) \ , \tag{40}$$

where $\ell\left(\boldsymbol{L}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{H}\right) = \mu\|\boldsymbol{s}\|_1 + (\boldsymbol{\alpha} - \boldsymbol{L}\boldsymbol{s})^{\top} \boldsymbol{H} (\boldsymbol{\alpha} - \boldsymbol{L}\boldsymbol{s})$, and $\boldsymbol{L}$ can be built incrementally in practice (see [35] for details). Critically, this online approximation removes the dependence of Equation 36 on the numbers of training samples and tasks, making it scalable for lifelong learning, and provides guarantees on its convergence with equivalent performance to the full multi-task objective [35]. Note that the $\boldsymbol{s}^{(t)}$ coefficients are only updated while training on task t and otherwise remain fixed.

Algorithm 6 ELIRL (k, λ, μ)

$\boldsymbol{L} \leftarrow \text{RandomMatrix}_{d,k}$
while some task $\mathcal{T}^{(t)}$ is available **do**
  $\mathcal{Z}^{(t)} \leftarrow \text{getExampleTrajectories}(\mathcal{T}^{(t)})$
  $\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{H}^{(t)} \leftarrow \text{inverseReinforcementLearner}(\mathcal{Z}^{(t)})$
  $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} (\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s})^{\top} \boldsymbol{H}^{(t)} (\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}) + \mu\|\boldsymbol{s}\|_1$
  $\boldsymbol{L} \leftarrow \text{update}L(\boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{H}^{(t)}, \lambda)$

This process yields the estimated reward function as $\boldsymbol{r}_{s_i}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}\mathbf{x}_{s_i}$. We can then solve the now-complete MDP for the optimal policy using standard RL. The complete ELIRL algorithm is given

as Algorithm 6. ELIRL can either support a common feature space across tasks, or can support different feature spaces across tasks by making use of prior work in autonomous cross-domain transfer [101].

Improving performance on earlier tasks As ELIRL is trained over multiple IRL tasks, it gradually refines the shared knowledge in $\boldsymbol{L}$. Since each reward function's parameters are modeled as $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, subsequent changes to $\boldsymbol{L}$ after training on task t can affect $\boldsymbol{\theta}^{(t)}$. Typically, this process improves performance in lifelong learning [35], but it might occasionally decrease performance through negative transfer, due to the ELIRL simplifications restricting that $\boldsymbol{s}^{(t)}$ is fixed except when training on task t. To prevent this problem, we introduce a novel technique. Whenever ELIRL is tested on a task t, it can either directly use the $\boldsymbol{\theta}^{(t)}$ vector obtained from $\boldsymbol{L}\boldsymbol{s}^{(t)}$, or optionally repeat the optimization step for $\boldsymbol{s}^{(t)}$ in Equation 39 to account for potential major changes in the $\boldsymbol{L}$ matrix since the last update to $\boldsymbol{s}^{(t)}$. This latter optional step only involves running an instance of the LASSO, which is highly efficient. Critically, it does not require either re-running MaxEnt or recomputing the Hessian, since the optimization is always done around the optimal single-task parameters, $\boldsymbol{\alpha}^{(t)}$. Consequently, ELIRL can pay a small cost to do this optimization when it is faced with performing on a previous task, but it gains potentially improved performance on that task by benefiting from up-to-date knowledge in $\boldsymbol{L}$, as shown in our results.

3.4.6 Task-agnostic lifelong learning using high-level shared sets of features (SHELS).

Representing concepts as exclusive sets of higher level features It is well-established in developmental psychology that children characterize concepts as collections of essential features, known as schemas [102]. As an example, a child living with a dalmatian dog may learn to characterize it by its four-legs, fur, tail, black and white spots, and medium size. These features constitute the child's dog schema. Critically, schemas can share features (enabling transfer between concepts), but are differentiated by their unique *sets* of essential features [103]. The child may also have a horse schema that shares the four-legs, fur, and tail features, but is larger. When the child first encounters a cow, it may differentiate it as a new animal, since it has never before seen a large animal with four-legs, spots, and a tail. It is the exclusive *set* of essential features that enables the child to both recognize concepts and detect that the cow is OOD. The child may mistakenly call the cow a horse, using their closest matching schema. With their parent's correction, the child can create a new cow schema to accommodate the new information. Thus, both features and schemas are continually acquired and refined over time.

From a mathematical standpoint, schemas represent a collection of exclusive sets of features. Precisely, we can state:

Definition 1 (Exclusive Sets) *Two sets $\mathcal{S}, \mathcal{T}$ are exclusive if and only if they each contain unique elements:* $\mathsf{exclusive}(\mathcal{S}, \mathcal{T}) \iff \mathcal{S} - \mathcal{T} \neq \emptyset \wedge \mathcal{T} - \mathcal{S} \neq \emptyset$. (Note that this definition inherently precludes subset relationships and prevents $\mathcal{S}$ and $\mathcal{T}$ from being empty.)

Definition 2 (Collection of Exclusive Sets) *A collection of sets $\mathcal{C}$ is exclusive if and only if every pair of sets within that collection is exclusive:* $\mathsf{exclusive}(\mathcal{C}) \iff \forall \mathcal{S}, \mathcal{T} \in \mathcal{C} \;\; \mathcal{S} \neq \mathcal{T} \implies \mathsf{exclusive}(\mathcal{S}, \mathcal{T})$.

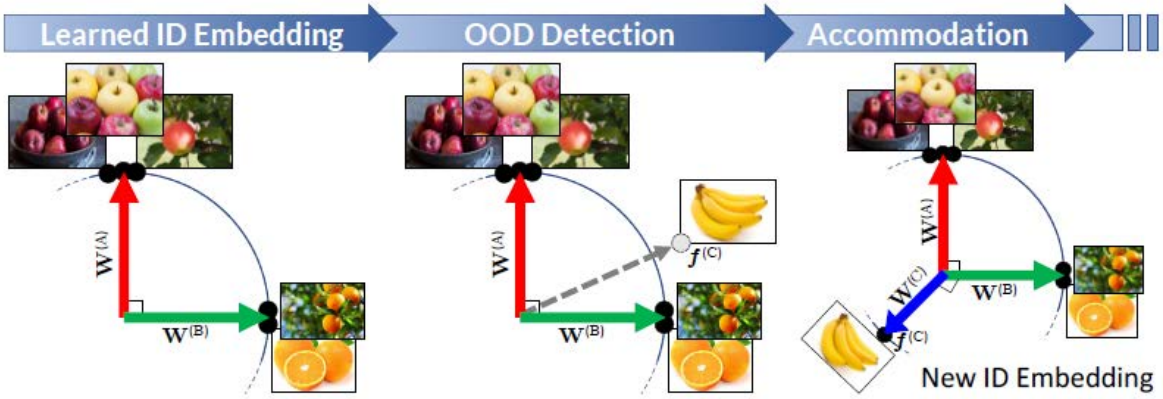


Figure 22: Representing Data with Exclusive Sets of Features through Orthogonal Embeddings to Enable Novelty Detection

To manifest the idea of schemas in a representational space for machine learning, we embed each data instance $\boldsymbol{x} \in \mathbb{R}^d$ as a set of derived features $\boldsymbol{f} = g(\boldsymbol{x}) \in \mathbb{R}^k$. Our goal is for the non-zero entries of the embeddings $\boldsymbol{f}$ of each class to form a collection of exclusive sets. *Orthogonality* between the vectors in the resulting vector space captures this notion of exclusive feature sets. In particular, a set of orthogonal feature embeddings $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_C \in \mathbb{R}^k$ for each of C classes forms a collection of exclusive sets. In fact, orthogonality is more stringent than exclusivity, since non-orthogonal features can still constitute exclusive sets. However, our work shows empirically that orthogonal feature embeddings are sufficient for both characterizing classes and enabling OOD detection.

To impose orthogonality, we employ cosine normalization [104] and use cosine scores in the loss function when learning. This embeds the in-distribution (ID) classes along the surface of the unit ball, spacing them apart. Intuitively, this maximizes the chance that a new OOD class would have an embedding with low cosine similarity with the other classes, facilitating OOD detection. A continual learner utilizing this technique can then adapt its embedding to preserve orthogonality when accommodating the new class, permitting future OOD detection of other new classes. Figure 22 illustrates this process, which we develop into a complete algorithm below.

Our approach trains DNNs that have 1) high-level feature sets that are exclusive to each class and 2) low-level features that are shared among classes. We call this the *SHELS* representation—a Sparse High-level-Exclusive Low-level-Shared representation. SHELS representations promote effective class discrimination and form capacity-efficient models, enabling both novelty detection and accommodation. Learning features this way differs substantially from typical DNN learning, which does not optimize for network efficiency and therefore exploits the full representational capacity, leading to representations that are redundant, correlated, and sensitive to the input distribution. Our approach learns SHELS representations using a combination of sparsity regularization and cosine normalization, enabling DNNs to learn features that are non-correlated, essential, and (at higher levels) exclusive to each class. We exploit this representation to detect new classes as novel activation patterns of high-level features, enabling OOD detection. We then accommodate the necessary information for new classes into the DNN by identifying and updating only those nodes that are unimportant to previously learnt classes, thereby mitigating catastrophic forgetting for continual learning.

Notation For a neural network of L hidden layers, let $\mathbf{W}^l$ denote the weight matrix for layer $l \in 1, \ldots, L$, with all weight matrices collected together into a tensor of parameters $\mathbf{W}$. The vector

Figure 23: Types of Exclusivity in SHELS: Exclusivity Encouraging Upper-Level Nodes to Choose Exclusive Sets of Lower Level Nodes (left), Group Sparsity Eliminates Redundant Neurons (right)

of outgoing weights from node n_g in layer l, which we denote a *group*, is given by $\mathbf{W}^l_{[g,:]}$, while the vector of incoming weights to node n_i is given by $\mathbf{W}^l_{[:,i]}$. $\Omega(\mathbf{W}^l)$ is a regularization term on the network weights at layer l and $\rho(n)$ is the average activation of a node n over a dataset. The features produced by layer l of the network on input $\boldsymbol{x}$ are represented by $\boldsymbol{f}^l(\boldsymbol{x})$ and $\boldsymbol{f}_n(\boldsymbol{x})$ denotes the feature activation at node n. We use $c \in 1, \ldots, C$ to denote seen
ID classes and $D_{1:C}$ to denote the dataset over all seen classes. In continual learning formulations, we use $\mathbf{W}^{(c)}$ to denote the weight tensor trained up to the c-th class.

Exclusive high-level features To impose exclusivity in the high-level features (Figure 23-left), we employ cosine normalization in the last layer of the neural network, in place of the usual dot product. Typical classification networks predict class scores for a given input $\boldsymbol{x}$ by applying the softmax function to the output of the last linear layer: $\mathrm{softmax}(\mathbf{W}^L \boldsymbol{f}^{L-1}(\boldsymbol{x}) + \boldsymbol{b}^L)$. We replace this linear transformation with the cosine of the angle between the weights $\mathbf{W}^L_{[:,c]}$ for class c and the features $\boldsymbol{f}^{L-1}(\boldsymbol{x})$ (i.e., the cosine similarity) to compute class scores:

$$\cos^{(c)}(\mathbf{W}, \boldsymbol{x}) = \frac{\mathbf{W}^{L\top}_{[:,c]} \boldsymbol{f}^{L-1}(\boldsymbol{x})}{\|\mathbf{W}^L_{[:,c]}\| \|\boldsymbol{f}^{L-1}(\boldsymbol{x})\|} \quad . \tag{41}$$

Using the raw cosine similarity scores as the class scores encourages the weights of the last layer to be sensitive to *which* features are activated instead of the degree of the feature activations. Therefore, training the network to optimize for the raw cosine scores results in orthogonal—and consequently exclusive—high-level features per class.

Shared lower level features Sparse regularizers are primarily used to reduce the complexity of DNNs by zeroing out certain weights. However, we can also use sparse regularizers to enable continual learning. For class-incremental continual learning, we need the ability to accommodate new information into the network without disrupting any knowledge learnt for previous classes. We accomplish this by creating sparsity in the network during training, allowing us to maintain unused nodes for learning future classes. We impose such sparsity in a structured fashion using the group Lasso regularizer defined by [105] and adapt it to DNNs [106, 107], promoting the removal of redundancies by eliminating correlated features while also encouraging the sharing of features:

$$\Omega_G(\mathbf{W}^l) = \sum \left\|\mathbf{W}^l_g\right\|_2 = \sum_g \sqrt{\sum_i \left(\mathbf{W}^l_{[g,i]}\right)^2} \quad . \tag{42}$$

Group Lasso promotes inter-group sparsity; that is, it eliminates groups (nodes) in the lower level that are redundant and are not shared among multiple high-level groups, as illustrated in Figure 23-right.

Algorithm 7 Novelty Detection Algorithm

1: `initTraining`$(D_{1:C}, \mu, \alpha, T)$:
2: Randomly initialize model $\mathcal{M}$ parameters, $\mathbf{W}$
3: Define loss $\mathcal{L}(\mathbf{W})$ as in Equation 43
4: **for** $t = 1, \ldots, T$ **do**
5: $\quad \mathbf{W}_t \leftarrow \mathbf{W}_{t-1} - \nabla\mathcal{L}(\mathbf{W}_{t-1})$
6: $\boldsymbol{th} \leftarrow$ `computeTh`$(\mathcal{M}, D_{1:C})$
7: **Output:** $\mathcal{M}$, $\boldsymbol{th}$
8:
9: `detect`$(X, \mathcal{M})$:
10: $\boldsymbol{s} \leftarrow$ `computeScore`$(X, \mathcal{M})$
11: $s^* \leftarrow \max(\boldsymbol{s})$
12: $c^* \leftarrow \arg\max(\boldsymbol{s})$
13: **if** $s^* > \boldsymbol{th}_{c^*}$ **then**
14: $\quad$ **Output:** class c^*
15: **else**
16: $\quad$ **Output:** $novel$

Layered sparse representation learning To learn SHELS representations, we combine group sparsity at different layers with cosine normalization at the last layer of the network by optimizing for the following loss:

$$\mathcal{L}(\mathbf{W}) = \underbrace{\frac{1}{|D_{1:C}|}\sum_{c=1}^{C}\sum_{k=1}^{|D_c|} -\log\Big(\cos^{(c)}(\mathbf{W}, \boldsymbol{x}_k)\Big)}_{\mathcal{L}_{CE}} + \alpha\sum_{l}\Big(1-\mu^{(l)}\Big)\underbrace{\Big(\sum_{g}\big\|\mathbf{W}^l_{[g,:]}\big\|_2\Big)}_{\Omega_G} \quad , \tag{43}$$

where $\mathcal{L}_{CE}$ is the standard cross-entropy loss applied to the cosine similarity scores of Equation 41 and Ω_G is the group sparsity regularizer. The parameter α controls the amount of regularization, and by setting $\mu^{(l)} = \frac{l-1}{L-1}$ we can interpolate between more sharing at the lower layers and less sharing (and therefore more exclusivity) at the higher layers, with $\mu^{(1)} = 0$ and $\mu^{(L)} = 1$ at the outermost layers.

Novelty detection With the learnt model composed of SHELS features, we exploit the exclusivity in the high-level feature sets with respect to each known class to detect novel classes. Since each class is represented by an exclusive set of high-level features (their *signature*), a novel class is detected when the high-level feature activations differ from the signatures of known (ID) classes. Concretely, we define the signature of class c as the mean high-level features over all seen data for that class.

Our novelty detector (Algorithm 7) first trains a model $\mathcal{M}$ for T epochs to learn SHELS features using the ID training data $D_{1:C}$ (lines 2-5). For an input $\boldsymbol{x}$, we compute a class similarity score $\boldsymbol{s}_c$ for each ID class $c \in 1, \ldots, C$ via:

$$\boldsymbol{s}_c(\boldsymbol{x}) = \mathbf{W}^{L\,\top}_{[:,c]}\boldsymbol{f}^{L-1}(\boldsymbol{x}) \quad , \tag{44}$$

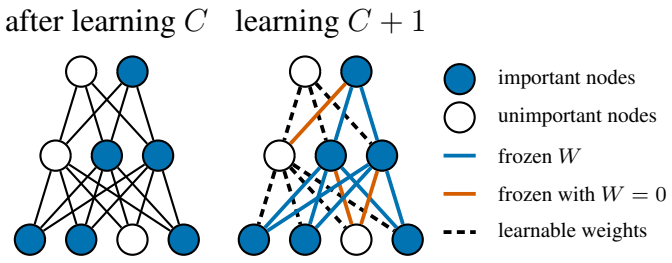


Figure 24: Novelty Accommodation via Selective Weight Updates

which measures similarity between the high-level features of the sample and the signature of known ID class c. We then compute thresholds $\boldsymbol{th}$ for each ID class c as the mean of $\boldsymbol{s}_c$ minus one standard deviation, computed over the training data of each class that the model correctly classifies (line 6). At inference time (lines 9-16), we compute the similarity score $\boldsymbol{s}(\boldsymbol{x})$ with respect to each class c and find the maximum over the classes' scores $s^*(\boldsymbol{x}) = \max \boldsymbol{s}(\boldsymbol{x})$ and the corresponding class $c^* = \arg\max \boldsymbol{s}(\boldsymbol{x})$. Then, $\boldsymbol{x}$ is predicted as belonging to class c^* if its similarity score $s^*(\boldsymbol{x})$ is greater than the threshold for the class, $\boldsymbol{th}_{c^*}$; otherwise, it is identified as *novel*.

Novelty accommodation Following the detection of a novel class, our framework updates the SHELS representation to accommodate it, which involves learning new knowledge relevant to the novel class while preserving previously learnt knowledge to prevent catastrophic forgetting [108]. Since group sparsity has the effect of eliminating certain nodes from the network, these unimportant nodes remain available for incorporating new knowledge. We measure a node's importance as the average activation of the node over the entire training data; [109] use a similar method to compute node importance. Precisely, we measure the importance of a node n up to class C as:

$$\rho^{(C)}(n) = \frac{1}{|D_{1:C}|} \sum_{k=1}^{|D_{1:C}|} \boldsymbol{f}_n(\boldsymbol{x}_k) \quad . \tag{45}$$

Our approach focuses the learning of the new class on unimportant nodes, while ensuring that important nodes are not modified. In particular, we target two types of modifications that could affect important nodes: *model drift* and *negative knowledge transfer*. Model drift occurs when the incoming weights of important nodes are changed during the learning of the new class. To avoid model drift, we penalize updates to these incoming weights. On the other hand, negative knowledge transfer occurs when an unimportant node is updated and used as input to an important node. To avoid negative knowledge transfer, we set the weights connecting unimportant nodes (i.e., $\rho^{(C)}(n) = 0$) to important nodes to zero and restrict their updates, thereby stabilizing important nodes. The combined approach for avoiding forgetting is illustrated in Figure 24.

To selectively update weights, we add a weight penalty regularizer Ω_{WP} to Equation 43 to penalize the incoming weights $\mathbf{W}_{[:,i]}$ of node n_i from being updated, weighted by the node's importance $\rho^{(C)}(n_i)$. This gives us the loss:

$$\mathcal{L}(\mathbf{W}^{(C+1)}) = \mathcal{L}_{CE} + \alpha \sum_l \sum_g \left((1-\mu) \left\| \mathbf{W}^{(C+1),l}_{[g,:]} \right\|_2 \right) + \beta \sum_l \sum_i \underbrace{\rho^{(C)}(n_i) \left\| \mathbf{W}^{(C+1),l}_{[:,i]} - \mathbf{W}^{(C),l}_{[:,i]} \right\|_2}_{\Omega_{WP}} \quad . \tag{46}$$

Algorithm 8 Novelty Accommodation Algorithm

1: $\rho^{(C)}(n) \leftarrow$ `computeNodeImp` $(D_{1:C}, \mathcal{M})$
2: `accomm`$(D_{C+1}, \mathcal{M}, \alpha, \beta, \mu, T, \rho^{(C)}(n))$:
3: Define loss $\mathcal{L}(\mathbf{W}^{(C+1)}, \mathbf{W}^{(C)})$ as in Equation 46
4: **for** $t = 1, \ldots, T$ **do**
5: $\quad \mathbf{W}_t^{(C+1)} \leftarrow \mathbf{W}_{t-1}^{(C+1)} - \nabla\mathcal{L}(\mathbf{W}_{t-1}^{(C+1)}, \mathbf{W}^{(C)})$
6: **Output:** $\mathcal{M}$

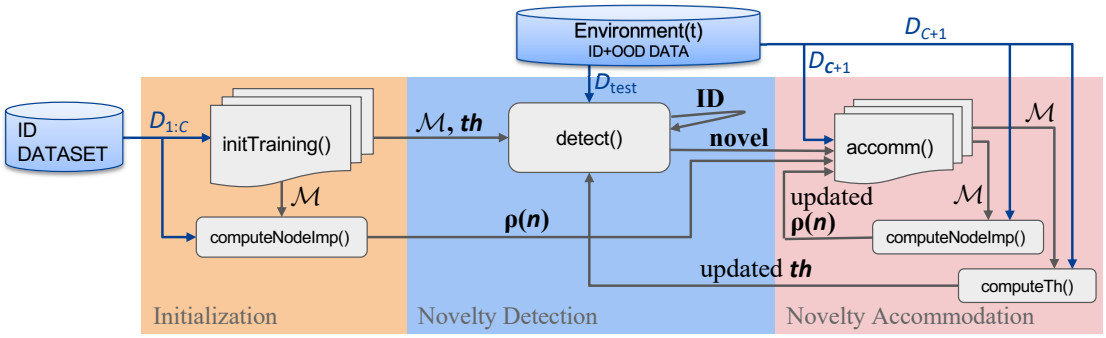
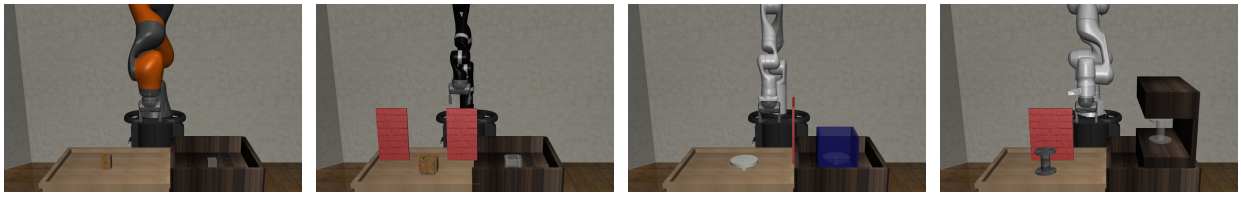


Figure 25: SHELS Approach to Novelty Detection and Accommodation without Class Boundaries

This ensures that the new class is primarily learnt in unimportant nodes, without modifying important nodes. The group sparsity and cosine normalization terms further ensure the ability to detect and accommodate future novel classes.

Our novelty accommodation method (Algorithm 8) first computes the node importance $\rho^{(C)}(n)$ for ID classes of the SHELS model $\mathcal{M}$ trained on dataset $D_{1:C}$ (line 1). Then, `accomm()` uses $\rho^{(C)}(n)$ to accommodate the novel class by updating $\mathcal{M}$ over T epochs, optimizing Equation 46 on data D_{C+1} (lines 3-6).

Novelty detection and accommodation for continual learning Our continual learning agent first learns the SHELS model $\mathcal{M}$ over the ID data $D_{1:C}$, and computes the thresholds $\boldsymbol{th}$ and the node importance values $\rho(n)$. The model is then deployed to a detection phase, using our novelty detection approach to perform inference on unseen data D_{test} in a dynamic environment, which may contain ID or OOD data. If it detects D_{test} as OOD with a confidence greater than a preset threshold, the agent switches to novelty accommodation. During accommodation, the agent receives additional data of the newly detected class D_{C+1} and accommodates the novel class by updating previously-unimportant nodes. Critically, the agent expands the representation learnt for the previous classes $1, \ldots, C$ to incorporate the new class $C + 1$ while maintaining orthogonality, ensuring that future classes can still be detected as OOD, as illustrated in Figure 22. Once the new class has been accommodated, the agent switches back to deployment mode, and the process repeats.



⟨**IIWA, box, no_obstacle, pick-and-place**⟩ ⟨**Jaco, hollow_box, object_door, push**⟩ ⟨**Gen3, plate, goal_wall, trash_can**⟩ ⟨**Panda, dumbbell, object_wall, shelf**⟩

Figure 26: Initial Conditions of Four CompoSuite Tasks

3.4.7 Benchmarking Compositional Multi-task and Lifelong Reinforcement Learning.

We developed CompoSuite—a simulated robotic manipulation benchmark designed to study the ability of RL algorithms to learn functional decompositions of the solutions to the tasks, yet more broadly applicable to multi-task and continual RL. The key idea is to build the tasks compositionally, so that 1) we can create combinatorially many (distinct) tasks, and 2) tasks are explicitly compositionally related. Sampled tasks are illustrated in Figure 26.

Task design CompoSuite is implemented on top of `robosuite` [110], a framework for the design of new simulated robotics environments in MuJoCo [111]. Concretely, CompoSuite is built around four compositional axes, which represent common modules that typically make up robotic manipulation programming pipelines: object grasp-pose detection, obstacle avoidance, task planning, and low-level motor control. There are four elements of each type (i.e., for each axis), so that combining them yields a total of 256 tasks; this represents the largest discrete set of tasks in a multi-task RL benchmark to date, yet remains computationally feasible. Within each axis, elements are designed such that a policy that succeeds at one task is very unlikely to succeed at another task—and the optimal policy for one task is even less likely to be optimal for another. To focus on the property of compositionality, variations within each axis are discrete, such that an agent can not trivially interpolate between the elements within one axis. Each environment contains two bins: one for objects and one for targets. This standardization encourages the agents to find the commonalities between the tasks. The reward functions are crafted to facilitate learning each individual task.

Task components

- **Robots** As the first axis of CompoSuite, we use simulated versions of commercially available robotic manipulators: KUKA's `IIWA`, Kinova's `Jaco`, Franka's `Panda`, and Kinova's `Gen3`. These manipulators vary in sizes, kinematic configurations, and position and torque limits, leading to semantic discrepancies between their observations and actions that require the agent to specialize its control policy for each arm. Consequently, a policy that works on one robot arm cannot be directly applied to another arm. To ensure compatibility with existing multi-task RL methods, we use arms with seven degrees of freedom (7-DoF). All arms use the Rethink Robotics two-finger gripper to manipulate objects.

- **Objects** We next consider four objects of distinct shapes that require orthogonal grasping orientations. The `box` is a cuboid that can be picked up from the top. The `hollow_box` resembles an open package, with a size sufficiently large that the gripper cannot grasp it by

both sides like the box, and must instead grip one of its edges. The dumbbell is placed upright, and its weights are larger than the gripper, and so it can only be grasped horizontally by the bar. The plate's diameter is also greater than the gripper size and therefore can only be grasped horizontally by the edge.

- **Obstacles** The third axis of variation in CompoSuite is a set of four obstacles that block off distinct areas for trajectory planning. The object_wall is a brick wall placed between the robot and the object, while the object_door is a similarly placed doorway between two brick walls. These two obstacles require avoiding opposite regions of the space while reaching for the object. The goal_wall is also a brick wall, but is placed between the left and right bins, blocking the direct path to goal after grasping the object. Additionally, we consider tasks with no_obstacle.

- **Task objectives** The final compositional axis is a set of different task objectives, each of which requires a unique sequence of steps for successful completion. The objectives are to pick-and-place an object into the right bin, push the object from the left to the right bin, drop the object into a trash_can, and place the object on a shelf.

Thanks to combinatorial explosion, there are 256 possible combinations of these components, leading to a set of 256 highly varied tasks. By design, each task requires a unique policy, but we know exactly how tasks relate to one another, enabling researchers to extract insights about the kind of compositionality that deep RL methods exhibit.

Observation and action spaces The observation space is split into the following factors, tied to the task components described in the previous section:

- *Robot observation* The proprioceptive portion of the observation space includes the sine and cosine of the robot's joint positions, its joint velocities, end effector pose, finger positions, and finger velocities.

- *Object observation* The agent observes both the absolute position and orientation of the object in world coordinates, as well as its position and orientation with respect to the robot's end effector. Note that this observation deliberately does not give away any information that distinguishes objects from one another (e.g., their geometric properties).

- *Obstacle observation* The agent also observes the absolute and relative positions and orientations of the obstacles. Similarly, this does not give away what the free space of the environment is (e.g., object_wall and object_door are always placed in the same location, but they block off opposite parts of the space).

- *Goal observation* The agent is also given the absolute and relative position and orientation of the goal, as well as the relative position of the goal with respect to the object. However, for simplicity, pick-and-place, trash_can, and shelf tasks are considered solved at any arbitrary location in the target region (e.g., the right bin or the shelf).

- *Task observation* The agent may also be given access to a multi-hot indicator that identifies each of the components of the task (i.e., the robot, object, obstacle, and objective). This is used as a task descriptor for multi-task training.

The action space is eight-dimensional, with the first seven dimensions providing target joint angles. Under the hood, a proportional-derivative (PD) controller executes the motor commands that follow the joint positions provided by the agent. The eighth dimension is a binary action that indicates whether the gripper should be open or closed.

Reward functions While CompoSuite supports sparse rewards for successful completion, this leads to an extremely hard exploration problem. Consequently, to isolate the problem of multi-task compositional learning, we provide a crafted reward that encourages exploration in stages, such that each stage leads the agent to a state that is closer to task completion.

During the initial **reach** stage, the agent is rewarded for reducing the distance from the gripper to the object. This stage terminates once the agent **grasps** the object, which gives a binary reward. These two initial stages are common to all objectives. In all tasks except for `push`, the agent is next rewarded for **lifting** the object up to a given height. In the case of `shelf` tasks, the agent is then encouraged to **align** the gripper with the horizontal plane, facing the shelf. The next stage rewards the agent for **approaching** the right bin (or the goal, in `push` tasks) based on the horizontal distance. In `pick-and-place` tasks, the reward then encourages the agent to **lower** the object down to the bin. In `trash_can` tasks, the agent is instead rewarded for **dropping** the object while above the trash can with a binary reward.

The final stage is a binary **success** reward. `pick-and-place` tasks succeed if the object is in the bin and the robot is near the object; this latter constraint differentiates `pick-and-place` and `trash_can` tasks. `push` tasks are solved if the object is near the goal location. The agent succeeds on `trash_can` tasks if the object is inside the trashcan and the gripper is *not*. The success criterion for `shelf` tasks is that the object is on the shelf.

The maximum possible reward is $R = 1$ and is only attained upon successfully executing the task. Table 2 summarizes the stages of each task objective.

Table 2: Reward Stages Per Task Objective

Task	Stages
`pick-and-place`	reach $\rightarrow$ grasp $\rightarrow$ lift $\rightarrow$ approach $\rightarrow$ lower $\rightarrow$ success
`push`	reach $\rightarrow$ grasp $\rightarrow$ approach $\rightarrow$ success
`trash_can`	reach $\rightarrow$ grasp $\rightarrow$ lift $\rightarrow$ approach $\rightarrow$ drop $\rightarrow$ success
`shelf`	reach $\rightarrow$ grasp $\rightarrow$ lift $\rightarrow$ align $\rightarrow$ approach $\rightarrow$ success

Episode initialization and termination Upon initialization of each new episode, the graspable object is placed in a random location of the left bin. In tasks that contain an obstacle, the object's initial location is restricted to the regions of the space that would explicitly require the robot to circumvent the obstacle. The goal locations are initialized in the right bin, and the robot arm is initialized at a fixed position with the gripper facing downward. Sampled initial conditions are displayed in Figure 26.

Each episode terminates after $H = 500$ time steps. In addition, `push` tasks terminate if the robot lifts the object more than a set (small) threshold above the table, in order to avoid success by the robot executing a pick-and-place strategy.

Evaluation settings CompoSuite evaluates agents for training speed and final performance over a subset of *training* tasks, akin to training sets in supervised single-task settings. While this is a measure of training performance, it corresponds to the standard evaluation setting of the large majority of works in RL. After training, agents are evaluated on a *test* set of unseen tasks. Both of these evaluations explore the ability of agents to discover compositional properties of the tasks.

Metrics Agents are evaluated according to two metrics. For an agent evaluated over N tasks, with M evaluation trajectories for each task, each trajectory of length H, the average metrics are computed as follows:

Return The standard cumulative returns:

$$\overline{R} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{t=1}^{H} R_i(s_t, a_t) \ .$$

This is the usual evaluation criterion for RL works and directly relates to the optimization objective.

Success The per-task success rate:

$$\overline{S} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \max_{t \in [1,H]} \mathbb{1}[R_i(s_t, a_t) = 1] \ ,$$

where $\mathbb{1}$ is the indicator function. Note that a trajectory is successful if at *any* time the agent is in a success state.

Evaluation on training tasks The agent is first evaluated on the tasks that it trains on. An agent that is capable of extracting the compositional properties of the tasks should be able to achieve transfer across the tasks. Ideally, this transfer should translate to both faster convergence in terms of the number of samples required to learn, as well as higher final performance after convergence. In particular, agents in this setting should be compared against an equivalent single-task agent that uses the same training mechanism but does so individually on every task, without any notion of shared knowledge.

Evaluation on test tasks The key property that CompoSuite assesses is the ability of approaches to combine trained components in novel combinations to handle new tasks. Following [112], this can take the following two forms:

- **Zero-shot generalization with task descriptors** If the agent is given the multi-hot indicators as part of the observation, then it could (in principle) solve new, unseen tasks without any training on them. This would be possible only if the agent learns the compositional structure of the tasks and is able to combine its existing components into a solution to the new task. Intuitively, after learning 1) the `pick-and-place` task with the `box` object avoiding the `object_door` obstacle using the `IIWA` arm, and 2) the `push` task with the `plate` object avoiding the `object_wall` obstacle using the `Panda` arm, if the agent knows how each of the components relates to the overall task, it could for example swap the `IIWA` and `Panda` arms and solve the opposite tasks without any additional training.
- **Few-shot generalization without task descriptors** Alternatively, the agent might not be informed of which components make up the current task, and be required to discover this information through experience. The goal of the agent should then be to discover this information as rapidly as possible in order to solve the new task with little experience.

Access to state decomposition The modular architectures of [113] and [112] require knowledge about which components of the observation affect which parts of the architecture. While this information is readily available in CompoSuite, fair performance comparisons would require noting whether the agent is given this decomposition of the state space. Note that both zero-shot and few-shot settings could be targeted with or without the state decomposition.

Sample of training tasks Understanding the compositional capabilities of RL algorithms requires a careful study of the sample of combinations (i.e., tasks) that is provided to the agent for training. We propose the following evaluation settings:

- **Uniform sampling** In the simplest setting, the training tasks are sampled uniformly at random, and the agent is asked to generalize to all possible combinations of the seen components. The agent therefore must learn to combine its knowledge in different ways after having seen each component in various combinations.
- **Restricted sampling** In this much harder setting, the training is restricted to a single task for one of the components and many tasks for other components (e.g., in CompoSuite$\setminus$`IIWA`, the agent sees only one `IIWA` task and must generalize to all other `IIWA` tasks). This is akin to Experiment 3 in the work of [114], which demonstrated that this is an onerous problem even in the supervised setting. While a complete evaluation would require various choices of restricted arms, objects, obstacles, and objectives, as an initial step we propose four evaluation settings: CompoSuite$\setminus$`IIWA`, CompoSuite$\setminus$`hollow_box`, CompoSuite$\setminus$`object_wall`, and CompoSuite$\setminus$`pick-and-place`. These restricted elements were empirically observed to be easier to learn than others by the single-task agents during development. Restricting access to an "easy" element enables the zero-shot evaluation to focus on generalization, without conflating it with the difficulty of the task itself. As an exception, the CompoSuite$\setminus$`object_wall` setting was selected over CompoSuite$\setminus$`no_obstacle` because generalizing to the `no_obstacle` element is trivial from any other obstacle.
- **Smaller-scale benchmarks** While large benchmarks like CompoSuite are appealing for studying multi-task RL at scale, developing ideas in such large task sets is often (unfortunately) prohibitively time-consuming. Given the compositional nature of CompoSuite, it is straightforward to extract smaller-scale benchmarks that maintain the properties of the full-scale benchmark. For example, CompoSuite$\cap$`IIWA` considers only the 64 `IIWA` tasks. Interestingly, such reduced benchmarks permit studying the difficulty of generalization across certain axes (e.g., if an agent can transfer knowledge across objects but not across robots, then it would perform much better on CompoSuite$\cap$`IIWA` than on the full CompoSuite). Following the rationale of the restricted setting, we propose to evaluate agents on: CompoSuite$\cap$`IIWA`, CompoSuite$\cap$`hollow_box`, CompoSuite$\cap$`no_obstacle`, and CompoSuite$\cap$`pick-and-place`.

3.4.8 Multi-actor-attention-critic for multi-agent reinforcement learning.

We start by introducing the necessary notation and basic building blocks for our approach. We then describe our ideas in detail.

Markov Games We consider the framework of Markov Games [115], which is a multi-agent extension of Markov Decision Processes. They are defined by a set of states, S, action sets for each of N agents, $A_1, ..., A_N$, a state transition function, $T : S \times A_1 \times ... \times A_N \rightarrow P(S)$, which defines the probability distribution over possible next states, given the current state and actions for each agent, and a reward function for each agent that also depends on the global state and actions of all agents, $R_i : S \times A_1 \times ... \times A_N \rightarrow R$. We will specifically be considering a partially observable variant in which an agent, i receives an observation, o_i, which contains partial information from the global state, $s \in S$. Each agent learns a policy, $\pi_i : O_i \rightarrow P(A_i)$ which maps each agent's observation to a distribution over its set of actions. The agents aim to learn a policy that maximizes their expected discounted returns, $J_i(\pi_i) = \mathbb{E}_{a_1 \sim \pi_1, ..., a_N \sim \pi_N, s \sim T}[\sum_{t=0}^{\infty} \gamma^t r_{it}(s_t, a_{1t}, .., a_{Nt})]$, where $\gamma \in [0, 1]$ is the discount factor that determines how much the policy favors immediate reward over long-term gain.

Policy Gradients Policy gradient techniques [57, 56] aim to estimate the gradient of an agent's expected returns with respect to the parameters of its policy. This gradient estimate takes the following form:

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \log(\pi_\theta(a_t|s_t)) \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(s_{t'}, a_{t'}) \tag{47}$$

Actor-Critic and Soft Actor-Critic The term $\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(s_{t'}, a_{t'})$ in the policy gradient estimator leads to high variance, as these returns can vary drastically between episodes. Actor-critic methods [116] aim to ameliorate this issue by using a function approximation of the expected returns, and replacing the original return term in the policy gradient estimator with this function. One specific instance of actor-critic methods learns a function to estimate expected discounted returns, given a state and action, $Q_\psi(s_t, a_t) = \mathbb{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(s_{t'}, a_{t'})]$, learned through off-policy temporal-difference learning by minimizing the regression loss:

$$\begin{aligned} \mathcal{L}_Q(\psi) &= \mathbb{E}_{(s,a,r,s') \sim D} \left[(Q_\psi(s, a) - y)^2 \right] \\ \text{where } y &= r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(s')} \left[Q_{\bar{\psi}}(s', a') \right] \end{aligned} \tag{48}$$

where $Q_{\bar{\psi}}$ is the target Q-value function, which is simply an exponential moving average of the past Q-functions and D is a replay buffer that stores past experiences.

To encourage exploration and avoid converging to non-optimal deterministic policies, recent approaches of maximum entropy reinforcement learning learn a soft value function by modifying the policy gradient to incorporate an entropy term [117]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim D, a \sim \pi}[\nabla_\theta \log(\pi_\theta(a|s))(-\alpha \ \log(\pi_\theta(a|s)) \ + Q_\psi(s, a) - b(s))] \tag{49}$$

where $b(s)$ is a state-dependent baseline (for the Q-value function). The loss function for temporal-difference learning of the value function is also revised accordingly with a new target:

$$y = r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(s')}[Q_{\bar{\psi}}(s', a') - \alpha \ \log(\pi_{\bar{\theta}}(a'|s'))] \tag{50}$$

While an estimate of the value function $V_\phi(s)$ can be used a baseline, we provide an alternative that further reduces variance and addresses credit assignment in the multi-agent setting in section 3.4.8.

The main idea behind our multi-agent learning approach is to learn the critic for each agent by selectively paying attention to information from other agents. This is the same paradigm of training critics centrally (to overcome the challenge of non-stationary non-Markovian environments) and executing learned policies distributedly. Figure 27 illustrates the main components of our approach.

Attention The attention mechanism functions in a manner similar to a differentiable key-value memory model [118, 119]. Intuitively, each agent queries the other agents for information about their observations and actions and incorporates that information into the estimate of its value function. This paradigm was chosen, in contrast to other attention-based approaches, as it doesn't make any assumptions about the temporal or spatial locality of the inputs, as opposed to approaches taken in the natural language processing and computer vision fields.

To calculate the Q-value function $Q_i^{\psi}(o, a)$ for the agent i, the critic receives the observations, $o = (o_1, ..., o_N)$, and actions, $a = (a_1, ..., a_N)$, for all agents indexed by $i \in \{1 \dots N\}$. We represent the set of all agents *except* i as $\backslash i$ and we index this set with j. $Q_i^{\psi}(o, a)$ is a function of agent i's observation and action, as well as other agents' contributions:

$$Q_i^{\psi}(o, a) = f_i(g_i(o_i, a_i), x_i) \tag{51}$$

where f_i is a two-layer multi-layer perceptron (MLP), while g_i is a one-layer MLP embedding function. The contribution from other agents, x_i, is a weighted sum of each agent's value:

$$x_i = \sum_{j \neq i} \alpha_j v_j = \sum_{j \neq i} \alpha_j h(V g_j(o_j, a_j))$$

where the value, v_j is a function of agent j's embedding, encoded with an embedding function and then linearly transformed by a shared matrix V. h is an element-wise nonlinearity (we have used leaky ReLU).

The attention weight α_j compares the embedding e_j with $e_i = g_i(o_i, a_i)$, using a bilinear mapping (ie, the query-key system) and passes the similarity value between these two embeddings into a softmax

$$\alpha_j \propto \exp(e_j^{\top} W_k^{\top} W_q e_i) \tag{52}$$

where W_q transforms e_i into a "query" and W_k transforms e_j into a "key". The matching is then scaled by the dimensionality of these two matrices to prevent vanishing gradients [120].

In our experiments, we have used multiple attention heads [120]. In this case, each head, using a separate set of parameters (W_k, W_q, V), gives rise to an aggregated contribution from all other agents to the agent i and we simply concatenate the contributions from all heads as a single vector. Crucially, each head can focus on a different weighted mixture of agents.

Note that the weights for extracting selectors, keys, and values are shared across all agents, which encourages a common embedding space. The sharing of critic parameters between agents is possible, even in adversarial settings, because multi-agent value-function approximation is, essentially, a multi-task regression problem. This parameter sharing allows our method to learn effectively in environments where rewards for individual agents are different but share common features. This method can easily be extended to include additional information, beyond local observations and actions, at training time, including the global state if it is available, simply by adding additional encoders, e. (We do not consider this case in our experiments, however, as our approach is effective in combining local observations to predict expected returns in environments where the global state may not be available).

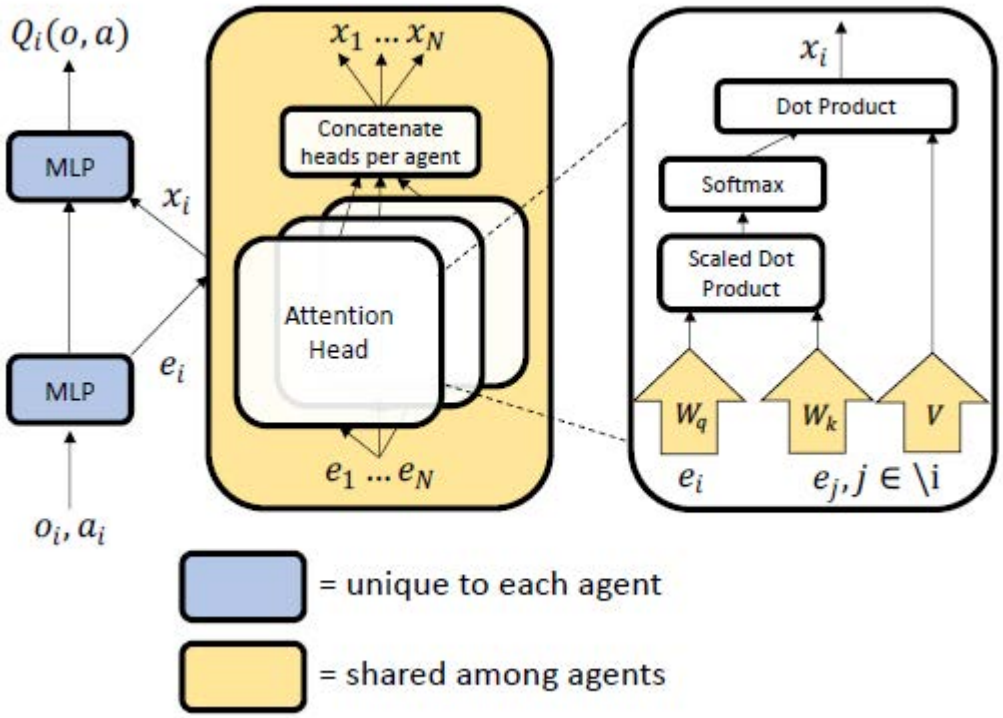


Figure 27: MAAC Model Architecture

Learning with Attentive Critics All critics are updated together to minimize a joint regression loss function, due to the parameter sharing:

$$\begin{aligned} \mathcal{L}_Q(\psi) &= \sum_{i=1}^{N} \mathbb{E}_{(o,a,r,o')\sim D}\left[(Q_i^{\psi}(o,a) - y_i)^2\right], \text{where} \\ y_i &= r_i + \gamma \mathbb{E}_{a'\sim\pi_{\bar{\theta}}(o')}[Q_i^{\bar{\psi}}(o',a') - \alpha\ \log(\pi_{\bar{\theta}_i}(a_i^{'}|o_i^{'}))] \end{aligned} \tag{53}$$

where $\bar{\psi}$ and $\bar{\theta}$ are the parameters of the target critics and target policies respectively. Note that Q_i^{ψ}, the action-value estimate for agent i, receives observations and actions for all agents. α is the temperature parameter determining the balance between maximizing entropy and rewards. The individual policies are updated by ascent with the following gradient:

$$\nabla_{\theta_i} J(\pi_\theta) = \mathbb{E}_{o\sim D, a\sim\pi}[\nabla_{\theta_i} \log(\pi_{\theta_i}(a_i|o_i))(-\alpha\ \log(\pi_{\theta_i}(a_i|o_i))\ + Q_i^{\psi}(o,a) - b(o,a_{\backslash i}))] \tag{54}$$

where $b(o, a_{\backslash i})$ is the multi-agent baseline used to calculate the advantage function decribed in the following section. Note that we are sampling all actions, a, from all agents' current policies in order to calculate the gradient estimate for agent i, unlike in the MADDPG algorithm [18], where the other agents' actions are sampled from the replay buffer, potentially causing overgeneralization where agents fail to coordinate based on their current policies [121].

Multi-Agent Advantage Function As shown in [19], an advantage function using a baseline that only marginalizes out the actions of the given agent from $Q_i^{\psi}(o,a)$, can help solve the multi-agent credit assignment problem. In other words, by comparing the value of a specific action to the value

of the average action for the agent, with all other agents fixed, we can learn whether said action will cause an increase in expected return or whether any increase in reward is attributed to the actions of other agents. The form of this advantage function is shown below:

$$\begin{aligned} A_i(o, a) &= Q_i^{\psi}(o, a) - b(o, a_{\setminus i})), \text{ where} \\ b(o, a_{\setminus i})) &= \mathbb{E}_{a_i \sim \pi_i(o_i)} \left[Q_i^{\psi}(o, (a_i, a_{\setminus i})) \right] \end{aligned} \tag{55}$$

Using our attention mechanism, we can implement a more general and flexible form of a multi-agent baseline that, unlike the advantage function proposed in [19], doesn't assume the same action space for each agent, doesn't require a global reward, and attends dynamically to other agents, as in our Q-function. This is made simple by the natural decomposition of an agents encoding, e_i, and the weighted sum of encodings of other agents, x_i, in our attention model.

Concretely, in the case of discrete policies, we can calculate our baseline in a single forward pass by outputting the expected return $Q_i(o, (a_i, a_{\setminus i}))$ for every possible action, $a_i \in A_i$, that agent i can take. We can then calculate the expectation exactly:

$$\mathbb{E}_{a_i \sim \pi_i(o_i)} \left[Q_i^{\psi}(o, (a_i, a_{\setminus i})) \right] = \sum_{a_i' \in A_i} \pi(a_i' | o_i) Q_i(o, (a_i', a_{\setminus i})) \tag{56}$$

In order to do so, we must remove a_i from the input of Q_i, and output a value for every action. We add an observation-encoder, $e_i = g_i^o(o_i)$, for each agent, using these encodings in place of the $e_i = g_i(o_i, a_i)$ described above, and modify f_i such that it outputs a value for each possible action, rather than the single input action. In the case of continuous policies, we can either estimate the above expectation by sampling from agent i's policy, or by learning a separate value head that only takes other agents' actions as input.

3.4.9 Randomized entity-wise factorization (REFIL) for multi-agent reinforcement learning.

In this work, we consider the *decentralized partially observable Markov decision process* (Dec-POMDP) [122] with entities [123], which describes fully cooperative multi-agent tasks.

Dec-POMDPs with Entities are described as tuples: $(\mathbf{S}, \mathbf{U}, \mathbf{O}, P, r, \mathcal{E}, \mathcal{A}, \Phi, \mu)$. $\mathcal{E}$ is the set of entities in the environment. Each entity e has a state representation s^e, and the global state is the set $\mathbf{s} = \{s^e | e \in \mathcal{E}\} \in \mathbf{S}$. Some entities can be agents $a \in \mathcal{A} \subseteq \mathcal{E}$. Non-agent entities are parts of the environment that are not controlled by learning policies (e.g., landmarks, obstacles, agents with fixed behavior). The state features of each entity consist of two parts: $s^e = [f^e, \phi^e]$ where f^e represents the description of an entity's current state (e.g., position, orientation, velocity, etc.) while $\phi^e \in \Phi$ represents the entity's type (e.g., outfield player, goalkeeper, etc.), of which there is a discrete set. An entity's type affects the state dynamics as well as the reward function and, importantly, it remains fixed for the duration of the entity's existence. Not all entities may be visible to each agent, so we define a binary observability mask: $\mu(s^a, s^e) \in \{1, 0\}$, where agents can always observe themselves $\mu(s^a, s^a) = 1, \forall a \in \mathcal{A}$. Thus, an agent's observation is defined as $o^a = \{s^e | \mu(s^a, s^e) = 1, e \in \mathcal{E}\} \in \mathbf{O}$. Each agent a can execute actions u^a, and the joint action of all agents is denoted $\mathbf{u} = \{u^a | a \in \mathcal{A}\} \in \mathbf{U}$. P is the state transition function which defines the

probability $P(\mathbf{s}'|\mathbf{s},\mathbf{u})$ and $r(\mathbf{s},\mathbf{u})$ is the reward function that maps the global state and joint actions to a single scalar reward.

Entities cannot be added during an episode, but they may become inactive (e.g., a unit dying in StarCraft) and no longer affect transitions and rewards. Since $\mathbf{s}$ and $\mathbf{u}$ are sets, their ordering does not matter, and our modeling construct should account for this (e.g., by modeling with permutation invariant/equivariant attention models [124]). In many domains, the set of entity types present $\{\phi^e|e\in\mathcal{E}\}$ is fixed. We are particularly interested in a multi-task setting where the quantity and types of entities are varied between tasks, as identifying patterns within sub-groups of entities is crucial to generalizing experience effectively in these cases.

Learning We aim to learn a set of policies that maximize expected discounted reward (returns). Q-learning is specifically concerned with learning an accurate action-value function Q^{tot} (defined below), and using this function to select the actions that maximize expected returns. The optimal Q-function for our setting is defined as:

$$\begin{aligned} &Q^{\text{tot}}(\mathbf{s},\mathbf{u}) := \\ &\mathbb{E}\Big[\textstyle\sum_{t=0}^{\infty}\gamma^t\, r(\mathbf{s}_t,\mathbf{u}_t)\;\Big|\; \begin{smallmatrix}\mathbf{s}_0=\mathbf{s},\ \mathbf{u}_0=\mathbf{u},\ \mathbf{s}_{t+1}\sim P(\cdot|\mathbf{s}_t,\mathbf{u}_t)\\ \mathbf{u}_{t+1}=\arg\max Q^{\text{tot}}(\mathbf{s}_{t+1},\cdot)\end{smallmatrix}\Big] \\ &\qquad = r(\mathbf{s},\mathbf{u}) + \gamma\,\mathbb{E}\Big[\max Q^{\text{tot}}(\mathbf{s}',\cdot)\;|\;{}_{s'\sim P(\cdot|\mathbf{s},\mathbf{u})}\Big] \end{aligned} \tag{57}$$

Partial observability is typically handled by using the history of actions and observations as a proxy for state, often processed by a recurrent neural network [125]: $Q^{\text{tot}}_\theta(\boldsymbol{\tau}_t,\mathbf{u}_t)\approx Q^{\text{tot}}(\mathbf{s}_t,\mathbf{u}_t)$ where the trajectory is $\tau^a_t := (o^a_0,u^a_0,\ldots,o^a_t)$ and $\boldsymbol{\tau}_t := \{\tau^a_t\}_{a\in\mathcal{A}}$. To learn the Q-function, deep reinforcement learning uses neural networks as function approximators trained to minimize the loss function:

$$\begin{aligned} \mathcal{L}_Q(\theta) &:= \mathbb{E}\Big[\Big(y^{\text{tot}}_t - Q^{\text{tot}}_\theta(\boldsymbol{\tau}_t,\mathbf{u}_t)\Big)^2\Big|(\boldsymbol{\tau}_t,\mathbf{u}_t,r_t,\boldsymbol{\tau}_{t+1})\sim\mathcal{D}\Big] \\ y^{\text{tot}}_t &:= r_t + \gamma Q^{\text{tot}}_{\bar\theta}\big(\boldsymbol{\tau}_{t+1},\arg\max Q^{\text{tot}}_\theta(\boldsymbol{\tau}_{t+1},\,\cdot\,)\big) \end{aligned} \tag{58}$$

where $\bar\theta$ are the parameters of a target network that is copied from θ periodically to improve stability [126] and $\mathcal{D}$ is a replay buffer [127] that stores transitions collected by an exploratory policy (typically ϵ-greedy). Double deep Q-learning [128] mitigates overestimation of the learned values by using actions that maximize Q^{tot}_θ as inputs for the target network $Q^{\text{tot}}_{\bar\theta}$.

Value Function Factorization Centralized training for decentralized execution (CTDE) has been a major focus in recent efforts in deep multi-agent RL [18, 19, 129, 130, 20]. Some methods achieve CTDE through factoring Q-functions into monotonic combinations of per-agent utilities, with each depending only on a single agent's history of actions and observations $Q^a(\tau^a,u^a)$. This factorization allows agents to independently maximize their local utility functions in a decentralized manner with their selected actions combining to form the optimal joint action. While factored value functions can only represent a limited subset of all possible value functions [131], they tend to perform better empirically than those that learn unfactored joint action value functions [132].

QMIX [130] improves over value decomposition networks (VDN) [129] by using a more expressive factorization than a summation of factors:

$$Q^{\text{tot}} = g\big(Q^1(\tau^1,u^1;\theta_Q),\ldots,Q^{|\mathcal{A}|}(\tau^{|\mathcal{A}|},u^{|\mathcal{A}|};\theta_Q);\theta_g\big)$$

The parameters of the monotonic mixing function θ_g are generated by a hyper-network [133] conditioning on the global state $\mathbf{s}$: $\theta_g = h(\mathbf{s}; \theta_h)$. Every state can therefore have a different mixing function; however, the mixing functions's monotonicity maintains decentralizability, as agents can greedily maximize Q^{tot} without communication. All parameters $\theta = \{\theta_Q, \theta_h\}$ are trained with the DQN loss of Equation 58.

Attention Mechanisms for MARL Attention models have recently generated intense interest due to their ability to incorporate information across large contexts, including in MARL [134, 20, 135]. Importantly for our purposes, they can process variable sized sets of fixed length vectors (in our case entities). At the core of these models is a parameterized transformation known as multi-head attention [120] that allows entities to selectively extract information from other entities based on their local context.

We define $\boldsymbol{X}$ as a matrix where each row corresponds to the state representation (or its transformation) of an entity. The global state $\mathbf{s}$ is represented in matrix form as $\boldsymbol{X}^{\mathcal{E}}$ where $\boldsymbol{X}_{e,*} = s^e$. Our models consist of entity-wise feedforward layers eFF($\boldsymbol{X}$), which apply an identical linear transformation to all input entities and multi-head attention layers MHA ($\mathcal{A}, \boldsymbol{X}, \boldsymbol{M}$), which integrate information across entities. The latter take three arguments: the set of agents $\mathcal{A}$ for which to compute an output vector, the matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{E}| \times d}$ where d is the dimensionality of the input representations, and a mask $\boldsymbol{M} \in \mathbb{R}^{|\mathcal{A}| \times |\mathcal{E}|}$. The layer outputs a matrix $\boldsymbol{H} \in \mathbb{R}^{|\mathcal{A}| \times h}$ where h is the hidden dimension of the layer. The row $\boldsymbol{H}_{a,*}$ corresponds to a weighted sum of linearly transformed representations from all entities selected by agent a. Importantly, if the entry of the mask $\boldsymbol{M}_{a,e} = 0$, then entity e's representation is not included in $\boldsymbol{H}_{a,*}$. Masking enables 1) decentralized execution by providing the mask $\boldsymbol{M}^{\mu}_{a,e} = \mu(s^a, s^e)$, such that agents can only see entities observable by them in the environment, and 2) "imagination" of the returns among sub-groups of entities. We integrate entity-wise feedforward layers and multi-head attention into QMIX in order to share a model across tasks where the number of agents and entities is variable and build our approach from there.

Attention Layers and Models Attention models have recently generated intense interest due to their ability to incorporate information across large contexts. Importantly for our purposes, they are able to process variable sized sets of inputs.

We now formally define the building blocks of our attention models. Given the input $\boldsymbol{X}$, a matrix where the rows correspond to entities, we define an entity-wise feedforward layer as a standard fully connected layer that operates independently and identically over entities:

$$\text{eFF}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}^\top, \boldsymbol{X} \in \mathbb{R}^{n^x \times d}, \boldsymbol{W} \in \mathbb{R}^{d \times h}, \boldsymbol{b} \in \mathbb{R}^h \tag{59}$$

Now, we specify the operation that defines an attention head, given the additional inputs of $\mathcal{S} \subseteq \mathbb{Z}^{[1,n^x]}$, a set of indices that selects which rows of the input $\boldsymbol{X}$ are used to compute queries such that $X_{\mathcal{S},*} \in \mathbb{R}^{|\mathcal{S}| \times d}$, and $\boldsymbol{M}$, a binary obserability mask specifying which entities each query entity can observe (i.e. $\boldsymbol{M}_{i,j} = 1$ when $i \in \mathcal{S}$ can incorporate information from $j \in \mathbb{Z}^{[1,n^x]}$ into its local context):

$$\text{Atten}(\mathcal{S}, \boldsymbol{X}, \boldsymbol{M}; \boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V) = \text{softmax}\left(\text{mask}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{h}}, \boldsymbol{M}\right)\right)\boldsymbol{V} \in \mathbb{R}^{|\mathcal{S}| \times h} \tag{60}$$

$$\boldsymbol{Q} = \boldsymbol{X}_{\mathcal{S},*}\boldsymbol{W}^Q, \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V, \quad \boldsymbol{M} \in \{0,1\}^{|\mathcal{S}| \times n^x}, \boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{d \times h} \tag{61}$$

The mask$(\boldsymbol{Y}, \boldsymbol{M})$ operation takes two equal sized matrices and fills the entries of $\boldsymbol{Y}$ with $-\infty$ in the indices where $\boldsymbol{M}$ is equal to 0. After the softmax, these entries become zero, thus preventing the attention mechanism from attending to specific entities. This masking procedure is used in our case to uphold partial observability, as well as to enable "imagining" the utility of actions within sub-groups of entities. Only one attention layer is permitted in the decentralized execution setting; otherwise information from unseen agents can be propagated through agents that are seen. $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V$ are all learnable parameters of this layer. Queries, $\boldsymbol{Q}$, can be thought of as vectors specifying the type of information that an entity would like to select from others, while keys, $\boldsymbol{K}$, can be thought of as specifying the type of information that an entity possesses, and finally, values, $\boldsymbol{V}$, hold the information that is actually shared with other entities.

We define multi-head-attention as the parallel computation of attention heads as such:

$$\text{MHA}\left(\mathcal{S}, \boldsymbol{X}, \boldsymbol{M}\right) = \text{concat}\left(\text{Atten}\left(\mathcal{S}, \boldsymbol{X}, \boldsymbol{M}; \boldsymbol{W}_j^Q, \boldsymbol{W}_j^K, \boldsymbol{W}_j^V\right), j \in \left(1 \ldots n^h\right)\right) \tag{62}$$

The size of the parameters of an attention layer does not depend on the number of input entities. Furthermore, we receive an output vector for each query vector.

Augmenting QMIX with Attention The standard QMIX algorithm relies on a fixed number of entities in three places: inputs of the agent-specific utility functions Q_a, inputs of the hypernetwork, and the number of utilities entering the mixing network, which must correspond the output of the hypernetwork since it generates the parameters of the mixing network. QMIX uses multi-layer perceptrons for which all these quantities have to be of fixed size. In order to adapt QMIX to the variable agent quantity setting, such that we can apply a single model across all tasks, we require components that accept variable sized sets of entities as inputs. By utilizing attention mechanisms, we can design components that are no longer dependent on a fixed number of entities taken as input. We define the following inputs: $\boldsymbol{X}_{ei}^{\mathcal{E}} := s_i^e, 1 \leq i \leq d, e \in \mathcal{E}; \boldsymbol{M}_{ae}^{\mu} := \mu(s^a, s^e), a \in \mathcal{A}, e \in \mathcal{E}$. The matrix $\boldsymbol{X}^{\mathcal{E}}$ is the global state $\mathbf{s}$ reshaped into a matrix with a row for each entity, and $\boldsymbol{M}^{\mu}$ is a binary observability matrix which enables decentralized execution, determining which entities are visible to each agent.

While the standard agent utility functions map a flat observation, whose size depends on the number of entities in the environment, to a utility for each action, our attention-utility functions can take in a variable sized set of entities and return a utility for each action. The attention layer output for agent a is computed as MHA $(\{a\}, \boldsymbol{X}, \boldsymbol{M}^{\mu})$, where $\boldsymbol{X}$ is an row-wise transformation of $\boldsymbol{X}^{\mathcal{E}}$ (e.g., an entity-wise feedforward layer). If agents share parameters, the layer can be computed in parallel for all agents by providing $\mathcal{A}$ instead of $\{a\}$, which we do in practice.

Another challenge in devising a QMIX algorithm for variable agent quantities is to adapt the hypernetworks that generate weights for the mixing network. Since the mixing network takes in utilities from each agent, we must generate feedforward mixing network parameters that change in size depending on the number of agents present, while incorporating global state information. Conveniently, the number of output vectors of a MHA layer depends on the cardinality of input set $\mathcal{S}$ and we can therefore generate mixing parameters of the correct size by using $\mathcal{S} = \mathcal{A}$ and concatenating the vectors to form a matrix with one dimension size depending on the number of agents and the other depending on the number of hidden dimensions. Attention-based QMIX (QMIX (Attention)) trains these models using the standard DQN loss.

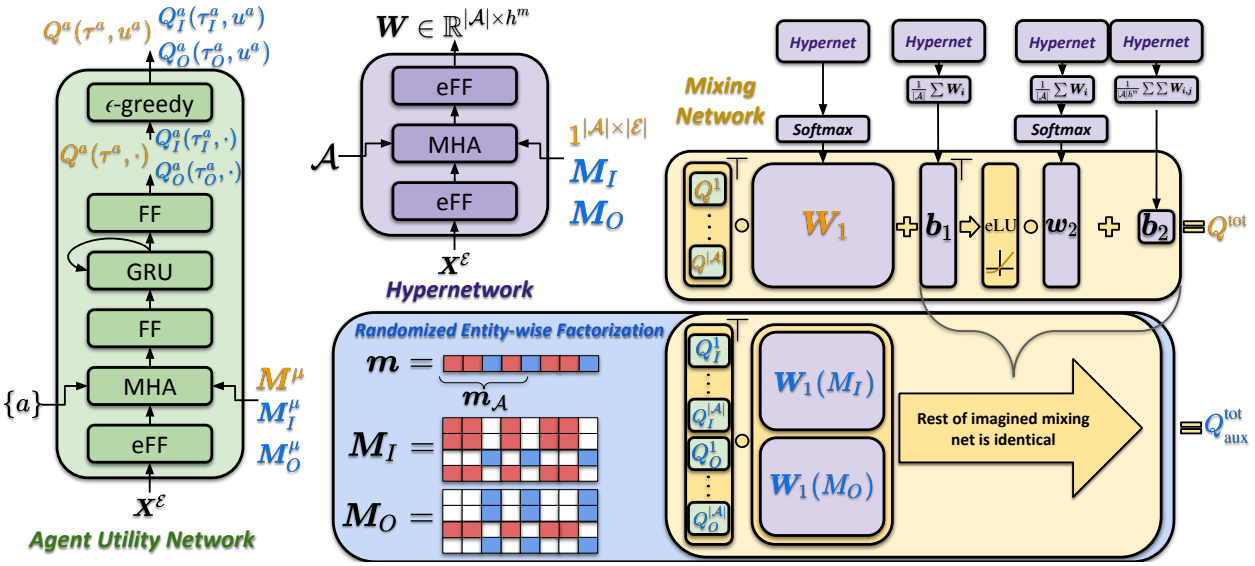


Figure 28: REFIL Model Architecture

Our two layer mixing network requires the following parameters to be generated: $\boldsymbol{W}_1 \in \mathbb{R}^{+(|\mathcal{A}| \times h^m)}$, $\boldsymbol{b}_1 \in \mathbb{R}^{h^m}$, $\boldsymbol{w}_2 \in \mathbb{R}^{+(h^m)}$, $b_2 \in \mathbb{R}$, where h^m is the hidden dimension of the mixing network and $|\mathcal{A}|$ is the set of agents.

Note from Equation 60 that the output size of the layer is dependent on the size of the query set. As such, using attention layers, we can generate a matrix of size $|\mathcal{A}| \times h^m$, by specifying the set of agents, $\mathcal{A}$, as the set of queries $\mathcal{S}$ from Equation 60. We do not need observability masking since hypernetworks are only used during training and can be fully centralized. For each of the four components of the mixing network ($\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{w}_2, b_2$), we introduce a hypernetwork that generates parameters of the correct size. Thus, for the parameters that are vectors ($\boldsymbol{b}_1$ and $\boldsymbol{w}_2$), we average the matrix generated by the attention layer across the $|\mathcal{A}|$ sized dimension, and for b_2, we average all elements. This procedure enables the dynamic generation of mixing networks whose input size varies with the number of agents. Assuming $\boldsymbol{q} = [Q^1(\tau^1, u^1), \ldots, Q^n(\tau^n, u^n)]$, then Q^{tot} is computed as:

$$Q^{\text{tot}}(\mathbf{s}, \tau, \mathbf{u}) = \sigma((\boldsymbol{q}^\top \boldsymbol{W}_1) + \boldsymbol{b}_1^\top)\boldsymbol{w}_2 + b_2 \tag{63}$$

where σ is an ELU nonlinearity [136].

Randomized Entity-wise Factorization for Imagined Learning We now propose **R**andomized **E**ntity-wise **F**actorization for **I**magined **L**earning (**REFIL**). We observe that common patterns often emerge in sub-groups of entities within complex multi-agent tasks and hypothesize that learning to predict agents' utilities within sub-groups of entities is a strong inductive bias that allows models to share information more freely across tasks. We instantiate our approach by constructing an estimate of the value function from factors based on randomized sub-groups, sharing parameters with the full value function, and training this factorized version of the value function as an auxiliary objective.

Random Partitioning Given an episode sampled from a replay buffer, we first randomly partition all entities in $\mathcal{E}$ into two disjunct groups, held fixed for the episode. We denote the partition by

a random binary[3] vector $\boldsymbol{m} \in \{0,1\}^{|\mathcal{E}|}$. m_e indicates whether entity e is in the first group. The negation $\neg m_e$ represents whether e is in the second group. The subset of all agents is denoted $\boldsymbol{m}_{\mathcal{A}} := [m_a]_{a\in\mathcal{A}}$. With these vectors, we construct binary attention masks $\boldsymbol{M}_I$ and $\boldsymbol{M}_O$:

$$\boldsymbol{M}_I := \boldsymbol{m}_{\mathcal{A}}\,\boldsymbol{m}^\top \vee \neg\boldsymbol{m}_{\mathcal{A}}\neg\boldsymbol{m}^\top, \boldsymbol{M}_O := \neg\boldsymbol{M}_I. \tag{64}$$

where $\boldsymbol{M}_I^\mu[a,e]$ indicates whether agent a and entity e are in the same group, and $\boldsymbol{M}_O^\mu[a,e]$ indicates the opposite. They are further combined with a partial observability mask $\boldsymbol{M}^\mu$, which is provided by the environment, to generate the final attention masks

$$\boldsymbol{M}_I^\mu := \boldsymbol{M}^\mu \wedge \boldsymbol{M}_I\,, \boldsymbol{M}_O^\mu := \boldsymbol{M}^\mu \wedge \boldsymbol{M}_O \tag{65}$$

These matrices are of size $|\mathcal{A}| \times |\mathcal{E}|$ and will be used by the multi-head attention layers to constrain which entities can be observed by agents.

Counterfactual Reasoning Given an *imagined* partition $\boldsymbol{m}$, an agent a can examine its history of observations and actions and reason *counterfactually* what its utility would be had it *solely* observed the entities in its group. We call this quantity *in-group utility* and denote it by $Q_I^a(\tau_I^a, u^a; \theta_Q)$. In order to account for the potential interactions with entities outside of the agents group, we calculate an *out-group utility*: $Q_O^a(\tau_O^a, u^a; \theta_Q)$. Note that the real and imagined utilities share the same parameters θ_Q, allowing us to leverage imagined experience to improve utility prediction in real scenarios and vice versa. Breaking the fully observed utilities Q^a into these randomized sub-group factors is akin to breaking an image into cut-outs of the comprising entities. While the "images" (i.e. states) from each task are a unique set, it's likely that the pieces comprising them share similarities.

Since we do not know the returns within the imagined sub-groups, we must ground our predictions in the observed returns. Just as QMIX learns a value function with n factors (Q^a for each agent), we learn an imagined value function with $2n$ factors (Q_I^a and Q_O^a for each agent) that estimates the same value:

$$\begin{aligned} Q^{\text{tot}} &= g\big(Q^1, \ldots, Q^{|\mathcal{A}|}; h(\mathbf{s}; \theta_h, \boldsymbol{M})\big) \\ &\approx Q^{\text{tot}}_{\text{aux}} = g\big(Q_I^1, \ldots, Q_I^{|\mathcal{A}|}, Q_O^1, \ldots, Q_O^{|\mathcal{A}|}; \\ &\qquad h(\mathbf{s}; \theta_h, \boldsymbol{M}_I), h(\mathbf{s}; \theta_h, \boldsymbol{M}_O)\big) \end{aligned} \tag{66}$$

Where $g(\cdot)$ are mixing networks whose parameters are generated by hypernetworks $h(\mathbf{s}; \theta_h, \boldsymbol{M})$. This network's first layer typically takes n inputs, one for each agent. Since we have $2n$ factors, we simply concatenate two generated versions of the input layer (using $\boldsymbol{M}_I$ and $\boldsymbol{M}_O$). We then apply the network to the concatenated utilities $Q_I^a(\tau_I^a, u^a)$ and $Q_O^a(\tau_O^a, u^a)$ of all agents a, to compute the predicted value $Q^{\text{tot}}_{\text{aux}}$. This procedure is visualized in Figure 28.

Importantly, since the mixing network is generated by the *full state context*, our model can weight factors contextually. For example, if the agent a's sampled sub-group contains all relevant information to compute its utility such that $Q_I^a \approx Q^a$, then the mixing networks can weight Q_I^a more heavily than Q_O^a. Otherwise, the networks learn to balance Q_I^a and Q_O^a for each agent, in order to estimate Q^{tot}. In this way, we can share knowledge in similar sub-group states across tasks while accounting for the differences in utility that result from the out-of-group context.

[3] We first draw $p \in (0,1)$ uniformly, followed by $|\mathcal{E}|$ independent draws from a Bernoulli(p) distribution. Partitioning into two groups induces a uniform distribution over all possible sub-groups.

Learning We now describe the overall learning objective of **REFIL**. To enforce Equation 66, we replace Q^{tot} in Equation 58 with $Q^{\text{tot}}_{\text{aux}}$, resulting a new loss $\mathcal{L}_{\text{aux}}$. We combine the standard QMIX loss (Eq. 58) $\mathcal{L}_Q$ with this auxiliary loss to form:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_Q + \lambda \mathbb{E}_{\boldsymbol{m}} \mathcal{L}_{\text{aux}} \tag{67}$$

where λ controls the tradeoff between the two losses. Note that we randomly partition in each episode, hence the expectation with respect to the partition vector $\boldsymbol{m}$. We emphasize that the sub-groups are *imagined*. While we compute $Q^{\text{tot}}_{\text{aux}}$ and its related quantities, we do not use them to select actions in Equation 58. Action selection is performed by each agent maximizing Q^a given their local observations. This greedy local action selection is guaranteed to maximize Q^{tot} due to the monotonic structure of the mixing network [130]. Moreover, our auxiliary objective is only used in training, and execution in the environment does not use random factorization. Treating random factorization as an auxiliary task, rather than as a representational constraint, allows us to retain the expressivity of QMIX value functions (without sub-group factorization) while exploiting the existence of shared sub-group states across tasks.

Implementation Details The model architecture is shown in Figure 28. "Imagination" can be implemented efficiently using attention masks. Specifically two additional passes through the network are needed, with $\boldsymbol{M}^{\mu}_{O}$ and $\boldsymbol{M}^{\mu}_{I}$ as masks instead of $\boldsymbol{M}^{\mu}$, per training step. These additional passes can be parallelized by computing all necessary quantities in one batch on GPU. It is feasible to split entities into an arbitrary number i of random sub-groups without using more computation by sampling several disjunct vectors $\boldsymbol{m}^i$ and combining them them in the same way as we combine $\boldsymbol{m}$ and $\neg\boldsymbol{m}$ in Equation 64 to form $\boldsymbol{M}_I$ and $\boldsymbol{M}_O$. Doing so could potentially bias agents towards considering patterns within smaller subsets of entities.

3.4.10 Solving stochastic traveling purchaser problem (STPP) for scavenger hunt service robot.

We denote a connected graph by $G = \{N, E\}$, where $N = \{n_1, n_2, \ldots, n_l\}$ is the node set and $E = \{e_{n,n'}\}_{n,n' \in N}$ is the cost set with any $e_{n,n'} \in E$ representing the minimal cost of traveling from node n to node n'. A stochastic traveling purchaser problem (STPP) consists of a graph $G = \{N, E\}$, an object set $O = \{o_1, \ldots, o_k\}$, a prior distribution D over N^k, and a starting node $n_0 \in N$. A STPP first generates a vector of object locations $X = (X_{o_1}, \ldots X_{o_k}) \in N^m$ from the prior distribution D, where X_{o_i} represents the location of object i for $i \in 1, \ldots, k$. The object locations X are unobservable to the searching agent. We denote t as the timestep and let $Y_t \in \{0,1\}^k$ be the task vector at timestep t , where $Y_{t,i} = 0$ if o_i has not been found and $Y_{t,i} = 1$ otherwise. An algorithm for solving a STPP problem starts from node n_0 and initial task vector $Y_0 = (0, 0, \ldots, 0)$. At each timestep $t \geq 1$, the searching agent takes an action by moving to node n_t, and incurring a cost e_{n_{t-1},n_t}. An agent's performance is evaluated by the total cost before all Y's are set to 1. Formally, the objective of a STPP problem is to minimize the cost as defined by Equation 68:

$$\begin{aligned} \min \quad & \sum_{t=1}^{t_1} e_{n_{t-1},n_t}, \text{ for } t_1 = \inf\{t > 0 : Y_{t,i} = 1, \forall i \in [k]\} \\ \text{given} \quad & G, D, O, n_0 \end{aligned} \tag{68}$$

Algorithm 9 describes a general framework for all the heuristic STPP algorithms in section 4. The algorithms differ in their implementation of the $choose\_next\_node()$ function on line 3. On line 5, the agent updates the task vector Y_t and a trajectory $\tau_t = (n_0, Y_0, e_{n_0,n_1}, n_1, Y_1, e_{n_1,n_2}, \ldots, n_{t-1}, Y_{t-1})$ up to time t after each timestep. A Bayesian posterior distribution $D(X|\tau_t)$ based on the prior distribution $D = D(X|\tau_0 = \emptyset)$ and the trajectory τ_t is computed for the algorithm to make a decision.

Algorithm 9 A General Framework for STPP Algorithms

Require: A graph $G = (N, E)$, a set of Objects O, a distribution D, start location node n_0

1: Initialize n_0, Y_0 and $\tau_0 = \emptyset$ with $t = 0$
2: **while** $Y_{t,i} \neq 0, \forall i \in [k]$ **do**
3: $n_{t+1} \leftarrow$ choose_next_node(G,D,O,n_t,τ_t)
4: Travel to n_{t+1}
5: Update D, Y_{t+1} and τ_{t+1} based on the occurrence of objects at n_{t+1}
6: $t = t + 1$

In the remaining part of this section, we present seven STPP-solving algorithms: one RL algorithm (DQN), one exhaustive search algorithm (Bayesian), three heuristic algorithms (Proximity, Probability, and Probability-Proximity), and two bounding algorithms (Offline Optimal, and Salesman). Complete pseudocode for all the algorithms is available at: https://github.com/utexas-bwi/scavenger_hunt_api/blob/master/PseudoCode.pdf

DQN Algorithm (RL) While the system is originally partially observable to the agent due to the unknown object locations, maintaining a dynamically updated Bayesian posterior distribution of the object positions as the state, leads to a regular MDP. Therefore, the algorithm considers the STPP problem as an MDP represented as a tuple (S, A, P, r, γ), with state-space $s \in S$: a state s is a set $\{D, n\}$, which includes the probability distribution D, combined with agent's current node n. The probability distribution is represented by a $l \times k$ matrix with element $p^n_{t_o}$ representing the probability of object o being located at node n; action-space $a \in A$: A is the same as the node set N with an action $a = n$ representing an action of travelling to node n; transition probability of states $P := p(s_{t+1}|s_t, a_t)$: a probability of transitioning from state s_t to the next state s_{t+1} given the action a_t; reward function $r := r(s, a)$: the negative distance of travelling from the agent's current node to the node specified by the action; a discount factor $\gamma = 0.95$ is set only for the purpose of training.

DQN with experience replay is implemented to train an agent with respect to just one specific hunt instance at a time. The Q-network is represented as a multi-layer perceptron with two 16-unit layers. Each of the layers is followed by a rectified linear unit (ReLu) activation function. The network takes as input a vector of probabilities of finding at least one object at each node, together with the shortest path costs between nodes. As we defined before, p^n_t is the probability of finding at least one object at node n and time step t with $n \in \{n_j\}$ for index $j = 1, 2, \ldots, l$ and we denote n_t as agent's current node, a vector $(p^{n_1}_t, p^{n_2}_t, \ldots, p^{n_l}_t, e_{n_1,n_t}, e_{n_2,n_t}, \ldots, e_{n_l,n_t})$ of length $2l$ is sent to the network as input. The agent's current node is apparent as being the only node with 0 travel cost. The network outputs the expected Q-values for each of the actions. The policy $\pi(a|s) := \arg\max_{a \in A} Q(s, a)$ is extracted by selecting the action that returns the highest Q value.

Exhaustive Bayesian Search (Exhaustive) This algorithm considers all possible paths, which are the sequences that contains all the unvisited nodes, and computes the expected cost of each path over all possible object locations $X \in N^k$. The path with the lowest expected cost over the distribution D is selected and the first node in the path is returned as the next node to visit.

Proximity-Based Search (Heuristic) This algorithm chooses the next node to visit by searching for the closest node that may potentially contain any unfound object. The number of objects that may appear at a node and probability of appearance are disregarded. We denote $p_{t_o}^n := D(X_o = n|\tau_t)$ as the posterior probability of object o located at node n based on the trajectory τ_t. The posterior probability incorporates past observations, including whether an object has already been found. Given the robot's current node n_t, the next node n_{t+1} is computed by the following equation:

$$n_{t+1} = \arg\min_{n \in N} \{e_{n_t,n} | \exists\ o \in O, p_o^n > 0\} \tag{69}$$

Probability-Based Search (Heuristic) This algorithm chooses the node with the greatest probability of finding at least one unfound object. The proximity of the location is disregarded, resulting in essentially the opposite of the proximity-based heuristic. If we denote p_t^n as the probability of finding at least one object at node n, the next node is computed by:

$$n_{t+1} = \arg\max_{n \in N} \{p_t^n\} \quad \text{and} \quad p_t^n = 1 - \prod_{o \in O} (1 - p_{t_o}^n) \tag{70}$$

Probability-Proximity Search (Heuristic) This algorithm is a combination of the proximity-based and probability-based heuristics. Each location node is scored according to the ratio of the probability of finding at least one object at that node to the distance from the current node to that node. Given the current node n_t and the probability of finding at least one object p_t^n, the next node is computed by the following equation:

$$n_{t+1} = \arg\max_{n \in N} \frac{p_t^n}{e_{n_t,n}} \tag{71}$$

The algorithm visits the node with the highest ratio. The rationale behind this scaling is to prioritize visiting nodes if the cost of traveling to them is worth the expected reward.

Salesman Search (Bounding) Serving as a naive baseline indicator on performance is a traveling salesman algorithm which visits all nodes that may contain objects, along the shortest path, while disregarding occupancy distributions [137]. The path is calculated once and not reevaluated during the search. This algorithm is intended as a lower bound on the performance of other heuristics, however, it is not a strict lower bound since for specific locations of objects, it can outperform other heuristics.

Offline Optimal Search (Bounding) The offline optimal search is informed offline of the objects locations X. It takes the shortest path between these locations. This is an (unachievable without omniscient knowledge of the object locations) upper bound on the performance of other algorithms, as it represents the fastest a STPP could possibly be completed.

3.4.11 Model-based Lifelong Reinforcement Learning with Bayesian Exploration (VBLRL).

Preliminaries RL is the problem of maximizing the long-term expected reward of an agent interacting with an environment [138]. We usually model the environment as a Markov Decision Process or *MDP* [139], described by a five tuple: $\langle S, A, R, T, \gamma \rangle$, where S is a finite set of states; A is a finite set of actions; $R : S \times A \mapsto [0, 1]$ is a reward function, with a lower and upper bound of 0 and 1; $T : S \times A \mapsto \Pr(S)$ is a transition function, with $T(s'|s, a)$ denoting the probability of arriving in state $s' \in S$ after executing action $a \in A$ in state s; and $\gamma \in [0, 1)$ is a discount factor, expressing the agent's preference for delayed over immediate rewards.

An MDP is a suitable model for the task facing a single agent. In the lifelong RL setting, the agent instead faces a series of tasks $m_1, ..., m_n$, each of which can be modeled as an MDP: $\langle S^{(i)}, A^{(i)}, R^{(i)}, T^{(i)}, \gamma^{(i)} \rangle$. For lifelong RL problems, the performance of a specific algorithm is usually evaluated based on both forward transfer and backward transfer results [140]:

- *Forward transfer*: the influence that learning task t has on the performance in future task $k \succ t$.
- *Backward transfer*: the influence that learning task t has on the performance in earlier tasks $k \prec t$.

A key question in the lifelong setting is how the series of task MDPs are related; we model the collection of tasks as a HiP-MDP, where a family of tasks is generated by varying a latent task parameter ω drawn for each task according to the world-model distribution P_Ω. Each setting of ω specifies a unique MDP, but the agent neither observes ω nor has access to the function that generates the task family. The dynamics $T(s'|s, a; \omega_i)$ and reward function $R(r|s, a; \omega_i)$ for task i then depend on $\omega_i \in \Omega$, which is fixed for the duration of the task. The tasks are i.i.d. sampled from a fixed distribution and arrive one at a time. Similar settings are commonly used in other lifelong RL papers [141].

Variational Bayesian Lifelong RL Our main intuition is that, if we initialize the task-specific distribution with a prior that is close to the true distribution, sample complexity will decrease accordingly. To scale our approach, we must find a efficient way to explicitly approximate these distributions. We propose a practical approximate algorithm, VBLRL, that uses neural networks and variational inference [142].

We choose Bayesian neural networks (BNN) to approximate the posterior. The intuition is that, in the context of stochastic outputs, BNNs naturally approximate the hierarchical Bayesian model since they also maintain a learnable distribution over their weights and biases [143, 144].

We use the uncertainty embedded in the weights and biases of networks to capture the epistemic uncertainty introduced by hidden parameters of different tasks, while we also set the outputs of the neural networks to be stochastic to capture the aleatory uncertainty within each specific task. In our case, the BNN weights and biases distribution $q(\omega; \phi)$ (a distribution over ω but parameterized by ϕ) can be modeled as fully factorized Gaussian distributions [145]:

$$q(\omega; \phi) = \prod_{j=1}^{|\Omega|} \mathcal{N}(\omega_j | \mu_j, \sigma_j^2), \tag{72}$$

where $\phi = \{\mu, \sigma\}$, and μ is the Gaussian's mean vector while σ is the covariance matrix diagonal.

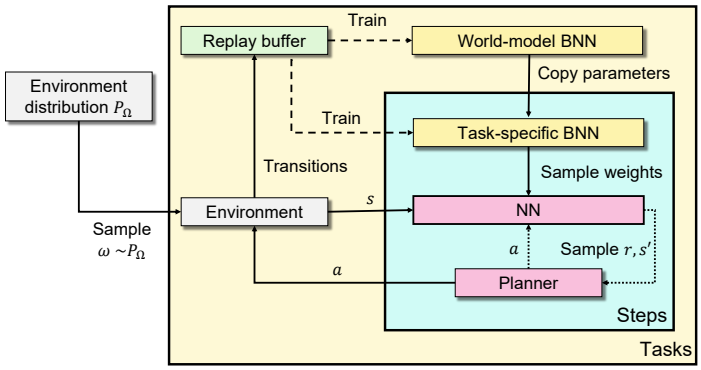


Figure 29: Variational Bayesian Lifelong Reinforcement Learning (VBLRL)

We maintain a world-model BNN across all the tasks and a task-specific BNN for each task. The input for all the BNNs is a state–action pair, and the output are the mean and variance of the prediction for reward and next state. Then, the posterior distribution over the model parameters can be computed leveraging variational lower bounds [146, 144]:

$$\phi_t = \arg\min_\phi \Big[D_{KL}[q(\omega;\phi)||p(\omega)] - \mathbb{E}_{\omega\sim q(\cdot;\phi)}[\log p(s^{t+1}, r^t|D^t, a^t;\omega)] \Big], \tag{73}$$

where $p(\omega)$ represents the fixed prior distribution of ω, also recall that $D^t = \{s^1, a^1, r^1, \cdots, s^t\}$. In VBLRL, $p(\omega)$ for the task-specific distribution is initialized by the world-model distribution, while $p(\omega)$ of the world-model distribution is simply set to a Gaussian. (That could be improved with more informed task family knowledge.) The second term on the right hand side can be approximated through $\frac{1}{N}\sum_{i=1}^{N} \log p(s^{t+1}, r^t|D^t, a^t;\omega_i)$ with N samples from $\omega_i \sim q(\phi)$. This optimization can be performed in parallel for each s, keeping ϕ_{t-1} fixed.

Once we have the world-model as well as the task-specific posterior model, we can do posterior sampling to drive Bayesian exploration. [147] and [148] already show the benefit of posterior sampling for improving sample efficiency in single-task RL and meta-RL settings. Here, instead of sampling from the posterior of value functions or latent context representations, we directly sample from the posterior of the transition model and choose the optimal action based on the transition samples. Then we use the collected samples to update the posterior and do sampling again. The agent thus continually do Bayesian exploration and acts more and more optimally as the posterior distribution narrows. We provide the framework of VBLRL in Figure 29. We employ our posterior knowledge models in the context of a model-based RL method. When encountering a new task, VBLRL first uses the model parameters (that is, $\{\mu, \sigma\}$ of weights and biases of BNN) from the general knowledge model to initialize the task-specific posterior network (directly copy parameters). We sample transitions from the task-specific posterior and select actions based on the generated transitions. The detailed algorithm is summarized in Algorithm 10. Specifically, for planning, at each step we begin by creating P particles from the current state $s^p_{\tau=t} = s_t \forall p$. Then, we sample N candidate action sequences $a_{t:t+T}$ from a learnable distribution. These two steps are the same as PETS [149]. Then we propagate the state–action pairs using the learned task-specific model $p_{m_i}(\cdot|s, a)$ (BNN) and use the cross entropy method [150] to update the sampling distribution to make the sampled action sequences close to previous action sequences that achieved high reward. We further calculate the cumulative reward estimated (via the learned model) for previously sampled

sequences and select the current action based on the mean of that distribution. By sampling from the task-specific posterior at each step, the agent explores in a temporally extended and diverse manner.

For each specific task, we use the task-specific BNN to plan during forward learning. They are updated with only the data from the current task thus avoid catastrophic forgetting in the forward transfer process. The world model is updated using the data from **all** the visited tasks.

The intuition is to guide the two posteriors to separately learn two categories of uncertainty within lifelong learning tasks. The world-model posterior captures the epistemic uncertainty of the general knowledge distribution (shared across all tasks controlled by the hidden parameters) via the internal variance of world-model BNN. As the learner is exposed to more and more tasks, the posterior should converge to P_Ω. The task-specific posterior captures the epistemic uncertainty of the current task m_i, which comes from the aleatory uncertainty of the world model when generating ω_i for a new task, via the internal variance of task-specific BNN. In general, we used the world-model BNN to encode P_Ω over a family of tasks, and the task-specific BNN to encode $P(\omega_i)$, which should peak at the specific true ω_i sampled for the current task.

Algorithm 10 Variational Bayesian Lifelong RL

Input: Initialize general knowledge(world) model $p_{wm}(\cdot|s, a; \omega_{wm})$, planning horizon T
for each task m_i from $i = 1, 2, 3, \cdots, M$ **do**
  Initialize task-specific model $p_{m_i}(\cdot|s, a; \omega_i)$ with parameters of general knowledge model p_{wm}
  for each episode **do**
    for Time $t = 0$ to TaskHorizon **do**
      Sample Actions $a_{t:t+T} \sim$ CEM($\cdot$)
      Propagate state particles s_τ^p with $p_{m_i}(s'|s, a; \omega_i)$
      Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} p_{m_i}(r|s, a; \omega_i)$
      Update CEM($\cdot$) distribution.
      Execute optimal actions $a^*_{t:t+T}$
    Add transitions to replay buffer D_{m_i}
    Update task-specific model according to Equation (73) given replay buffer D_{m_i}
    Update general knowledge model according to Equation (73) given replay buffers $\{D_{m_1}, \cdots, D_{m_i}\}$

Backward Transfer of Variational Bayesian Lifelong RL In our lifelong RL setting, the agent interacts with each task for only a limited number of episodes and the task-specific model stops learning when the next task is initiated. As a result, there may exist portions of the transition dynamics in which model uncertainty remains high. However, as the world-model posterior continues to train on new tasks, it gathers more experience in the whole state space and can provide improved guesses concerning the "unknown" transition dynamics, even for previously encountered tasks.

Intuitively, the performance of an agent on one task has the potential to be further improved (positive backward transfer) if there exists a sufficiently large set of state–action transition pairs of which the task-specific model's predictions are not confident due to lack of data.

In our algorithm, the aleatory uncertainty (irreducible chance in the outcome) is measured by the output variance of the prediction $\{\sigma_{r_\tau^p}, \sigma_{s_\tau^p}\}$, and the epistemic uncertainty (due to lack of experience)

corresponds to the uncertainty of the output mean and variance (see Definition 1 below). Thus, a straightforward method to improve a previously learned task-specific model is to find the predictions it needs to make that have high epistemic uncertainty, and replace them with the predictions from the world-model posterior, which has lower epistemic uncertainty. If we only consider reward prediction, the quantity for measuring whether a task-specific/world model is sufficiently confident is as follows.

Definition 1 For a given state–action pair (s, a), we define the confidence level c of the predictions (reward) $r_\tau^p(s, a)$ from a task-specific/world model as:

$$c = -\frac{\sum_{p=1}^{P}(\mu_{r_\tau^p} - \overline{\mu}_{r_\tau^p})^2}{P-1} - \alpha * \frac{\sum_{p=1}^{P}(\sigma_{r_\tau^p} - \overline{\sigma}_{r_\tau^p})^2}{P-1}, \tag{74}$$

where P is the number of particles and α is a hyperparameter controlling the scale of the second term. A similar definition applies to the task-specific/world model's next-state prediction. Intuitively, c measures the uncertainty of the output mean and variance for each dynamic prediction. During CEM planning, we propagate each pair of P state particles with different network parameters thanks to the usage of BNN, resulting in P different $\{\mu_{r_\tau^p}, \sigma_{r_\tau^p}\}$. Thus we can further calculate the confidence level of current predictions based on these P pairs of output mean and variance.

We show the detailed backward transfer algorithm in Algorithm 11. Compared with the forward training version, the agent will also calculate the confidence level of the task-specific and general-knowledge model for each particular transition and compare the value of them when predicting the state particles. If the task-specific model is not confident enough for this state–action pair (i.e. $c_{m_i} < c_{wm}$), we will use the world-model to do the predictions instead.

Algorithm 11 Variational Bayesian Lifelong RL (Backward transfer)

Input: Test task m_i, planning horizon T, task-specific model $p_{m_i}(s', r|s, a; \omega_i)$, general-knowledge model $p_{wm}(s', r|s, a; \omega_{wm})$
for Time $t = 0$ to TaskHorizon **do**
  for Trial $k = 1$ to K **do**
    Sample Actions $a_{t:t+T} \sim$ CEM$(\cdot)$
    for each action **do**
      Propagate state particles s_τ^p with $p_{m_i}(s', r|s, a)$
      Propagate state particles s_τ^p with $p_{wm}(s', r|s, a)$
      Compute confidence level c_{m_i} for task-specific model and c_{wm} for general-knowledge model (**Definition 4.5**)
      Choose the propagation results from the model with higher confidence level c
    Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r_\tau^p$
    Update CEM$(\cdot)$ distribution.
  Execute optimal actions $a_{t:t+T}^*$

3.4.12 Compositional Lifelong Reinforcement Learning.

The problem of lifelong functional composition in RL An RL problem $\mathcal{Z}$ is a composition of subproblems $F_1, F_2, \ldots$ if its optimal policy π^* can be constructed by combining solutions to

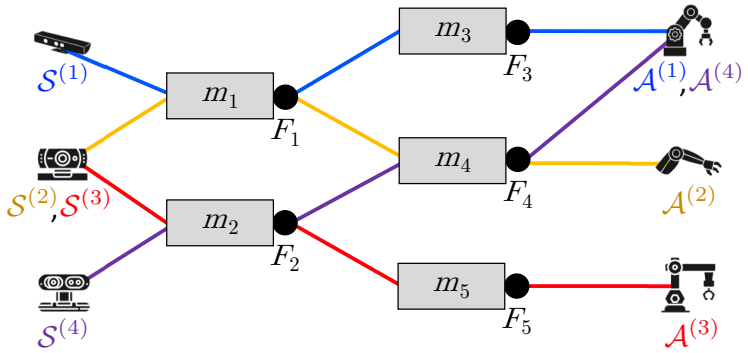


Figure 30: Compositional Reinforcement Learning Problem Graph

those subproblems: $\pi^*(\boldsymbol{x}) = m_1 \circ m_2 \circ \cdots$, where each $m_i \in \mathcal{M} : \mathcal{X}_i \mapsto \mathcal{Y}_i$ is the solution to the corresponding F_i. We first consider these subproblems from an intuitive perspective, and later define them precisely. In RL, each subproblem could involve pure sensing, pure acting, or a combination of both. For instance, in our earlier robotic manipulation example, the task can be decomposed into recognizing objects (sensing), detecting the target object and devising a plan to reach it (combined), and actuating the robot to grasp the object (acting).

In lifelong compositional RL, the agent will be faced with a sequence of MDPs $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T)}$ over its lifetime. We assume that all MDPs are compositions of different subsets from k shared subproblems $\mathcal{F} = \{F_1, \ldots, F_k\}$. The goal of the lifelong learner is to find the set of solutions to these subproblems as a set of modules $M = \{m_1, \ldots, m_k\}$, such that learning to solve a new problem reduces to finding how to combine these modules optimally. Each module can be viewed as a processing stage in a hierarchical processing pipeline, or equivalently as functions in a program, and the goal of the agent is to find the correct module to execute at each stage and the instantiation of that module (i.e., its parameters). We formalize the compositional RL problem as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (e.g., Figure 30) whose nodes are the subproblem solutions along with the state and action spaces: $\mathcal{V} = \mathcal{F} \bigcup \breve{\mathcal{X}} \bigcup \breve{\mathcal{U}}$, where $\breve{\mathcal{X}} = \text{unique}(\{\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(T)}\})$ and $\breve{\mathcal{U}} = \text{unique}(\{\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(T)}\})$ are the unique state and action spaces. Each subproblem F_i corresponds to a latent representational space $\mathcal{Y}_i$, generated by the corresponding module $m_i : \mathcal{X}_i \mapsto \mathcal{Y}_i$. For example, an object detection module could map an image to a segmentation mask, to be used by subsequent modules to make action decisions. Similarly, the $\mathcal{X}^{(t)}$'s and $\mathcal{U}^{(t)}$'s can serve as representation spaces $(\mathcal{X}_t, \mathcal{Y}_t)$.

Problem $\mathcal{Z}^{(t)}$ is then specified as a pair of state and action nodes $(\mathcal{X}^{(t)}, \mathcal{U}^{(t)})$ in the graph, and the goal is to find a path between those nodes corresponding to a policy $\pi^{(t)*}$ that maximizes $R^{(t)}$. More generally, the graph formalism allows for recurrent computations via walks with cycles, and parallel computations via concurrent multipaths; an extended definition of *multiwalks* trivially captures both settings. In this work, we consider the path formulation, and we restrict the number of edges in the graph by organizing the modules into layers. This formalism is related to that of [66] for the supervised compositional setting, but is adapted to RL. From this definition, we outline two concrete problem settings, based on the amount of information available to the agent. Both settings show how learning could benefit from compositional solutions.

- **Zero-shot generalization with full information** In some scenarios, the agent may have access to a task descriptor that encodes how the current task relates to others in terms of

their compositional structures. This descriptor might be sufficient to combine modules into a solution (i.e., zero-shot generalization), provided that the agent has learned to map the descriptors into a solution structure. The descriptor could take different forms, such as a multi-hot encoding of the various components, natural language, or highlighting the target objects in the input image. Our experiments study multi-hot descriptors as a means to provide the compositional structure. Formally, we assume that the descriptor is given as an external input $t \in \mathcal{T}$, and that there exists some function $s : \mathcal{T} \times \mathcal{M} \mapsto \Pi$ that can map this input and an optimal set of modules M into an optimal policy for the current task $s(t, M) = \pi^{(t)*}$. This would enable the agent to achieve *compositional* generalization: the ability to solve a new task entirely by reusing components learned from previous tasks.

- **Fast adaptation with restricted information** In other scenarios, the agent is not given the luxury of information about the compositional structure. This is common in RL, where the only supervisory signal is typically the reward. In this case, the agent would be incapable of zero-shot transfer. Instead, we measure generalization as its ability to *learn from limited experience* a task-specific function $s^{(t)} : \mathcal{M} \mapsto \Pi$ that combines existing modules into the optimal policy for the current task $s^{(t)}(M) = \pi^{(t)*}$. Intuitively, if the agent has accumulated a useful set of modules M, then we would expect it to be capable of quickly discovering how to combine and adapt them to solve new tasks.

Neural modular policy architecture We propose to handle modular RL problems via neural composition. Modular architectures have been used in the supervised setting to handle compositional problems [151, 65]. In RL, a few recent works have considered similar architectures, but without a substantial effort to study their applicability to truly compositional problems [152, 153]. One notable exception proposed a specialized modular architecture to handle multi-task, multi-robot problems [113]. We take inspiration from this latter architecture to design our own modular architecture for more general compositional RL.

Following the compositional assumptions, each neural module m_i in our architecture is in charge of solving one specific subproblem F_i (e.g., finding an object's grasping point in the robot tasks), such that there is a one-to-one and onto mapping from subproblems to modules. All tasks that require solving F_i will share m_i. To construct the network for a task, modules are chained in sequence, thereby replicating the graph structure depicted in Figure 30 with neural modules.

A typical modular architecture considers a pure chaining structure, in which the complete input is passed through a sequence of modules. Each module is required to not only process the information needed to solve its subproblem (e.g., the obstacle in the robot examples), but also to pass through information required by subsequent modules. Additionally, the chaining structure induces brittle dependencies among the modules, such that changes to the first module have cascading effects. While in MTL it is viable to learn such complex modules, in the lifelong setting the modules must generalize to unseen combinations with other modules after training on just a few tasks in sequence. One solution is to let each module m_i only receive information needed to solve its subproblem F_i, such that it need only output the solution to F_i. Our architecture assumes that the state can factor into module-specific components, such that each subproblem F_i requires only access to a subset of the state components and passes only the relevant subset to each module. For example, in the robotics domain, robot modules only receive as input the state components related to the robot state. Equivalently, each element of the state vector is treated as a variable and only the variables

Algorithm 12 Lifelong Compositional RL

$T \leftarrow 0$
loop
  $t \leftarrow \texttt{getTask}()$; $T \leftarrow T + 1$
  if $T \leq k$ // Initialize modules
    $\texttt{steps} \leftarrow 0$; $s_d \leftarrow T \; \forall d$
  else // Find module combination
    $s, \texttt{steps} \leftarrow \texttt{discreteSearch}()$
    $\texttt{bckModules} \leftarrow \texttt{clone}(M)$
  $\pi^{(t)}, Q^{(t)} \leftarrow \texttt{modularNets}(M, s)$
  // Online exploration
  while $\texttt{steps} < \texttt{onlineSteps}$
    $\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{s}' \leftarrow \texttt{rollouts}(\pi^{(t)}, \texttt{iterSteps})$
    $\texttt{buffer}[t].\texttt{push}(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{s}')$
    $\pi^{(t)}, Q^{(t)} \leftarrow \texttt{RLStep}(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{s}', \pi^{(t)}, Q^{(t)})$
    $\texttt{steps} \leftarrow \texttt{steps} + \texttt{iterSteps}$
  // Off-line comp. improvement
  if $T < \texttt{numModules}$
    $M \leftarrow \texttt{bckModules}$
  for $i = 1, \ldots, \texttt{offlineSteps}$
    for $t = 1, \ldots, \texttt{seenTasks}$
      $\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{s}' \leftarrow \texttt{buffer}[t].\texttt{sample}()$
      $\pi^{(t)} \leftarrow \arg\min_{\tilde{\pi}} \texttt{NLL}(\tilde{\pi}(\boldsymbol{x}), \boldsymbol{u})$
      $\boldsymbol{u}' \leftarrow \arg\max_{\tilde{\boldsymbol{u}}: \pi^{(t)}(\boldsymbol{s}', \tilde{\boldsymbol{u}}) > \tau} Q^{(t)}(\boldsymbol{s}', \tilde{\boldsymbol{u}})$
      $\texttt{targetQ} \leftarrow r + {Q^{(t)}}'(\boldsymbol{s}', \boldsymbol{u}')$
      $Q^{(t)} \leftarrow \arg\min_{\tilde{Q}} (\tilde{Q}(\boldsymbol{x}, \boldsymbol{u}) - \texttt{targetQ})$

necessary for solving each subproblem F_i are fed into m_i. This process requires only high-level information about the semantics of the state representation, similar to the architecture of [113].

At each depth d in our modular net, the agent has access to k_d modules to choose from. Each module is a small neural network that takes as input the module-specific state component along with the output of the module at the previous depth $d-1$. Note that this differs from gating networks in that the modular structure is fixed for each individual task instead of modulated by the state or input. The number of modular layers $d_{\max}$ is the number of subproblems that must be solved (e.g., $d_{\max} = 3$ for (1) grasping an object and (2) avoiding an obstacle with (3) a robot arm).

Sequential module learning Our method for lifelong learning of modular policies accelerates the learning of new tasks by leveraging existing modules, while simultaneously preventing the forgetting of knowledge stored in those modules. We achieve this by separating the learning into two main stages: an online stage in which the agent discovers which modules to use and explores the environment by modifying a copy of those modules, and an off-line stage in which the original modules are updated with data from the new task. The rationale is that, in the early stages of training, there are two conflicting goals: 1) flexibly adapting the module parameters to the current task to explore how to solve it and 2) keeping the module parameters as stable as possible to retain performance on earlier tasks. Our method (Algorithm 12) consists of the following stages.

- **Initialization** Finding a good set of initial modules is a major challenge. In order to achieve lifelong transfer to future tasks, we must start from a sufficiently strong and diverse set of modules. For this, we train each new task on disjoint sets of neural modules until all modules have been initialized. Intuitively, this corresponds to the assumption that the first tasks presented to the agent are made up of disjoint sets of components or subproblems, though we show empirically that this assumption is not necessary in practice.

- **Online training of new tasks** In most RL tasks, during the initial stages of training, the agent struggles to find relevant knowledge about the task. Lifelong RL methods typically ignore this, and incorporate new (likely incorrect) information into the shared knowledge all throughout training. Instead, we keep shared modules fixed during this online training phase, subdivided into two stages:
  - *Module selection* We consider two versions of our method. In one case, the choice of modules is given to the agent. In the other case, the agent must discover which modules are relevant for the current task. Since a diverse set of modules has already been initialized, this stage uses exhaustive search of the possible module combinations in terms of the reward they yield when combined.
  - *Exploration* Once the set of modules to use for the current task has been selected, the agent might be able to perform reasonably well on the task. However, especially on the earliest tasks, it is unlikely that modules that have never been combined will work together perfectly. Therefore, the agent might still need to explore the current task. In order to avoid catastrophic damage to existing neural components and at the same time enable full flexibility for exploring the current task, we execute this exploration via standard RL training on a *copy* of the shared modules.

  The rationale for executing selection and exploration separately is that we want the selected modules to be updated as little as possible in the next and final stage. If we were to instead jointly explore via module adaptation and search over module configurations, we are likely to find a solution that makes drastic modifications to the selected modules. If instead we restrict module selection to the fixed modules, then this stage is more likely to find a solution that requires little module adaptation.

- **Off-line incorporation of new knowledge via batch RL** While the exploration stage enables learning about the current task, this is typically insufficient for training a lifelong learner, since 1) we would need to store all copies of the modules in order to perform well on new tasks, and 2) the modules obtained from initialization are often suboptimal, limiting their potential for future transfer. For this reason, once the agent has been given enough experience on the current task, newly discovered knowledge is incorporated into the original shared modules. It is crucial that this accommodation step does not discard knowledge from earlier tasks, which is not only necessary for solving those earlier tasks (thereby avoiding catastrophic forgetting) but possibly also for future tasks (i.e., forward transfer). Drawing from the lifelong learning literature, we use experience replay over the previous tasks to ensure knowledge retention. However, while experience replay has been tremendously successful in the supervised setting, its success in RL has been limited to very short sequences of tasks [76, 154]. In part, this limited success stems from the fact that typical RL methods fail in training from fully off-line data, since the mismatch between the off-line data distribution and the data distribution

imposed by the updated policy tends to cause degrading performance. For this reason, we propose a novel method for experience replay based on recent batch RL techniques, designed precisely to avoid this issue. Concretely, at this stage the learner uses batch RL over the replayed data from all previous tasks as well as the current task, keeping the selection over components fixed and modifying only the shared modules. Note that this is a general solution that applies beyond compositional algorithms to any lifelong method that includes a stage of incorporating knowledge into a shared repository after online exploration.

We use *proximal policy optimization* (PPO) for online training and *batch-constrained Q learning* (BCQ) for batch RL. BCQ simultaneously trains an actor π to mimic the behavior in the data and a critic Q to maximize the values of actions that are likely under the actor π [155, 156]. In the lifelong setting, this latter constraint ensures that the Q-values are not over-estimated for states and actions that are not observed during the online stage. Note that other batch RL methods could be used instead.

3.4.13 Possibility Before Utility: Learning and Using Hierarchical Affordances (HAL).

Preliminaries Our setting involves learning behavior policies in *Markov Decision Processes* (MDP) using RL. An MDP is defined by the state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and state transition distribution $P : \mathcal{S} \times \mathcal{A} \to \triangle(\mathcal{S})$. The objective is to learn a policy, $\pi : \mathcal{S} \to \triangle(\mathcal{A})$, which selects actions that maximize expected future returns: $G(\pi) = \mathbb{E}\sum_{t=0}^{\infty} \gamma^t r_t \,|\, a_t \sim \pi, s_t \sim P$, where γ is the discount factor. [157] introduce the *options framework*, which flexibly models hierarchical abstractions with minimal modification to the RL paradigm. Each *option*, $o := \langle \mathcal{I}_o, \pi_o, \beta_o \rangle$, is defined by an initiation set, $\mathcal{I}_o \subseteq \mathcal{S}$, indicating where the option can be selected, the corresponding option policy, π_o, and the termination condition, $\beta_o : \mathcal{S}^+ \to [0, 1]$, indicating the probability of termination in each state. Options turn our typical MDP into a *semi-Markov decision process (SMDP)* since the state transition distribution is, in general, no longer dependent on the current state and action, but also the present option, which was decided in a previous time-step. The design of this framework allows options to be treated similarly to actions, except they may be executed across multiple time-steps, interrupted, composed, and learned as separate subpolicies.

Milestones and Hierarchical Affordances We consider tasks that can be decomposed into *subtasks*, each represented by a *milestone* symbol, $g \in \mathcal{G}$, where $\mathcal{G}$ is the set of symbols relevant to the task, and $|\mathcal{G}| = K$. For each subtask, we introduce a separate option, $\langle \mathcal{I}_g, \pi_g, \beta_g \rangle$, and we call π_g a *subpolicy*. At each time-step, in addition to the extrinsic reward signal provided by the environment to indicate success on the overall task, we have access to a *milestone signal*, which is a vector $\boldsymbol{b}^t$ where each element $b_g^t \in \{0, 1\}$ indicates whether $g \in \mathcal{G}$ was completed on time-step t. In our PASTA example, we might receive a milestone each time we cut a vegetable, make the sauce, cook the noodles, etc. Milestones serve two main purposes. Firstly, milestones function as option *subgoals* [157] that are in this work used to train each subpolicy (discussed in the next section). Secondly, each milestone represents the *intent* of its corresponding subtask—similar to the action intents introduced to learn action-level affordances in the work of [158]—which we use to learn *hierarchical* affordances (discussed later). In contrast to the standard options framework, primitive actions can only be executed as part of an option's subpolicy in our method. Generally, policies

trained solely over options have no guarantee of optimality [157], but we ensure the existence of an optimal solution by requiring $g_K \in \mathcal{G}$, where g_K is the task's final milestone (indicating task success). When $\mathcal{G} = \{g_k\}$, our setting is standard, flat RL. Each additional milestone g' added to $\mathcal{G}$ is useful as an intermediate signal so long as g' corresponds to a unique behavior necessary for achieving g_K.

Hierarchical affordances are defined over $\mathcal{G}$ in the following way. The vector $\boldsymbol{f}^s = f^*(s)$ of size K represents which milestones are *immediately* achievable from the present state s, without requiring the collection of any intermediate milestones, where f is a *hierarchical affordance classifier*, and f^* is the optimal one[4]. Formally, $f_g^s = 1$ if at time t_0 it is possible for future $b_g^T = 1$ without any $b_j^t = 1, j \neq g$, where $t_0 < t < T$. In PASTA, the milestone `mix cooked noodles and sauce` is not afforded at the beginning since `cook noodles` is required first. A successful policy trained within the vanilla options framework will *eventually* learn to execute options in contexts where they are most useful, regardless of each option's predefined initiation set. However, hierarchical affordances give us a principled way to directly adjust this set: for subtask g, we can set $\mathcal{I}_g = \{s \mid s \in \mathcal{S},\ f_g^s = 1\}$. One can think of hierarchical affordances as using milestones to impose a state-grounded subtask dependency structure on top of the options framework, which we can use to prune impossible subtasks. If $\mathcal{G} = \{g', g_K\}$, an affordance-aware agent with access to optimal $f^*(s)$ will never initiate subtask g_K from the beginning if g' is a necessary intermediate behavior.

Some logic-based RL approaches [160, 161] use *atomic propositions* as markers of subtask achievement to transition between contexts in a finite state machine. These approaches, and many other HRL works [162, 163] define subtask preconditions in terms of *other subtasks*. There are two forms of stochasticity that hierarchical affordances, by virtue of being grounded in the present state, can more naturally address than symbolically-defined dependencies. We can conceptualize potential agent trajectories as graphs where *nodes* represent the attainment of milestones, and *edges* are the segments between them. *Node stochasticity* is affordance-affecting randomness that occurs either when milestones are attained (e.g. receiving varying quantity of an item), or at the beginning of the episode (i.e. starting in different contexts). *Edge stochasticity* is when affordances change at *any time* within a segment. We treat edge stochasticity events as infrequent *exceptions* to the typical subtask dependency rules. For example, after `cook noodles` is complete, `mix cooked noodles and sauce` is afforded, even if the agent may eventually spill the noodles on the floor. By grounding these rules in the current state, an affordance-aware agent can detect and adapt to edge anomalies. In what follows, we describe in detail how hierarchical affordances are learned and used in stochastic environments where symbols alone would fail to reliably determine the current context.

Learning Controllers Like h-DQN [164], we use a *meta-controller* that selects the current subtask to attempt and a low-level *controller* which executes the subpolicy relevant to that subtask. The controller, $\pi : \mathcal{S} \times \mathcal{G} \to \triangle(\mathcal{A})$, selects low-level actions, $a \in \mathcal{A}$, given a state, $s \in \mathcal{S}$, and milestone, $g \in \mathcal{G}$, and aims to maximize the expected milestone signal rewards, b_g. Q-Learning [165] trains these controllers by learning an estimate of the optimal Q-function: $Q_{\text{c}}^*(s, a; g) = \max_\pi \mathbb{E}\sum_{t=0}^{\infty} \gamma^t b_g^t \mid s_0 = s, a_0 = a, a_t \sim \pi, s_t \sim P$ and deriving a policy from the Q-function as such: $\pi_g(a|s, g) = \mathbb{1}(a = \arg\max_{a'} Q_{\text{c}}(s, a'; g))$. Deep Q-Learning [166] estimates Q^* using deep neural networks. This Q-function is parameterized by $\theta = \{\theta_{\text{base}}, ..., \theta_g, ...\}$, where

[4]Unlike option *completion* predictions [159], affordances predict possibility of *success*.

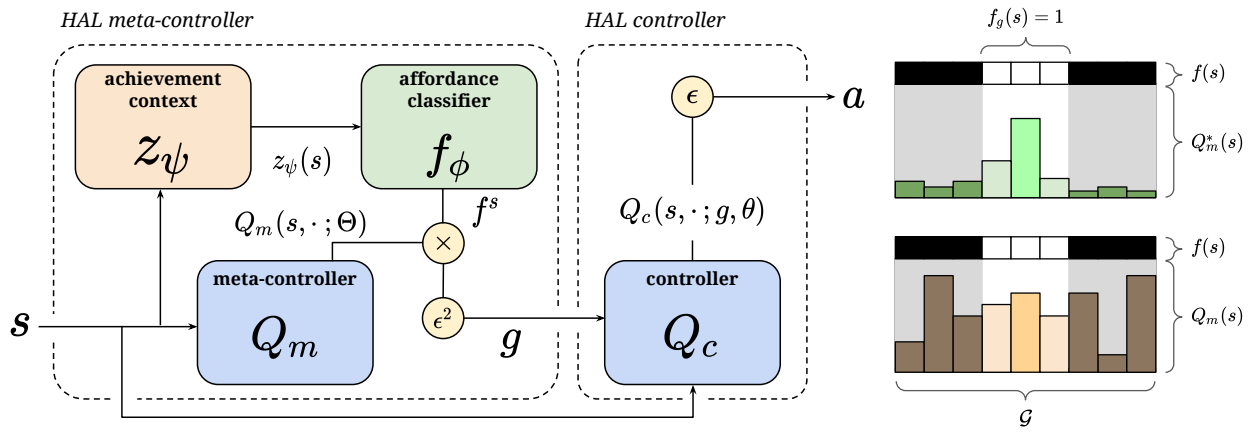


Figure 31: Left: Architecture Diagram for Complete HAL Method Right: Optimal (top) vs. Sub-optimal (bottom) Policy Behavior

θ_{base} is a set of shared base parameters and θ_g is a goal-specific head. It is updated via gradient descent on the following loss function, derived from the original Q-learning update:

$$\mathcal{L}_{Q_c} = \mathbb{E}(s_t, a_t, r_t, s_{t+1}, g_t) \sim D_c \left(Q_c(s_t, a_t; g_t, \theta) - b_g^t - \max_{a_{t+1}} Q_c(s_{t+1}, a_{t+1}; g_t, \bar{\theta}) \right)^2 \tag{75}$$

where D_c is a replay buffer that stores previously collected transitions, and $\bar{\theta}$ are the parameters of a periodically updated target network. Both of these components are included to avoid the instability associated with using function approximation in Q-Learning.

The meta-controller $\Pi : \mathcal{S} \to \triangle(\mathcal{G})$ aims to execute subtasks to maximize extrinsic rewards received by the environment. Again, we estimate a Q-function, this time over a dilated time scale (i.e. we allow the low-level controllers to run for multiple steps before choosing new goals): $Q^*_{\text{mc}}(s, g) = \mathbb{E} \sum_{t=0}^{N} r_t + \max_{g'} Q^*_{\text{mc}}(s_N, g') \mid s_0 = s, g_0 = g, a_t \sim \pi_g, s_t \sim P$, where N is the (variable) number of steps the option runs for. When collecting data in the environment, we add transitions $(s_t, g_t, \sum_t^{t+N} r_t, s_{t+N})$ to a separate meta-replay buffer, D_{mc}, used to train our meta-Q-function (parameterized by Θ) with a loss similar to Equation 75, but without any goal-conditioning. Next, we describe how hierarchical affordances are integrated into this training procedure.

Hierarchical Affordance Learning Now we introduce *Hierarchical Affordance Learning* (HAL), a method that addresses the challenges inherent to learning affordances over high-level subtasks, enabling more efficient learning in environments with complex subtask dependency structures. In typical HRL methods, if the meta-controller is yet to receive extrinsic reward from the environment, there will be no preference for selecting any subtask over the others. However, by restricting the selection of subtasks to ones that have proven merely to be *possible*, an agent can avoid wasting time attempting the impossible, and reach more fruitful subtasks faster. Suppose from experience, gained through random exploration, the agent achieves milestone g (where achievement means $b_g = 1$) very often from the initial state set $\mathcal{I}$, but never $j \in \mathcal{G}$, despite being able to achieve j in later states. With enough experience, the agent should become confident that j is not achievable without completing other milestones first, and should not bother selecting j from any $s \in \mathcal{I}$, while g,

and any others that *are* achievable from those states, should instead be considered. If we had access to an oracle function, $f^*(s)$, that accurately computes hierarchical affordances for our task, we could prune impossible subtasks by masking the otherwise uninformed policy with the affordance oracle output: $p(g|s) \propto f_g^s * \Pi_\Theta(g|s)$. In the following sections, we describe a method that can learn an approximate $f(s) \approx f^*(s)$ from experience and leverage it in real-time for more effective learning.

The false negative problem Recall that $f(s)$ outputs a vector $\boldsymbol{f}^s$, where each $\boldsymbol{f}_g^s$ indicates the possibility of collecting milestone g from state s without requiring intermediate milestones. To train each binary classification head, $f_g(s)$, we must somehow generate labeled data for each milestone. Suppose an option was initialized at time t_o, and in time T milestone g is received. If no others were received since the start of the option, we may assume that for any t where $t_o \leq t \leq T$, the collection of milestone g was afforded, so we can use the set of states $\{s_{t_o}, s_{t_o+1}, \ldots, s_T\}$ as positive (i.e. $f_g^*(s) = 1$) training examples for the affordance classifier. Even if g was not the intended milestone, we can still generate positive training examples for f_g in this way. If an option has failed to collect intended milestone g, either through timing out or collecting an unintended milestone, we might be tempted to use the states encountered during that option as negative examples (i.e. $f_g^*(s) = 0$). However, this occurrence can either be indicative of the states not affording g, or that the subpolicy corresponding to g is sub-optimal and has failed despite g being afforded. These false negatives are a problem for any approach requiring function approximation via neural networks, which are generally not robust to label noise [167]. It is particularly troublesome in our case since the noise is greater than the true signal when the subpolicies are under-trained.

Context learning Suppose we had access to an abstract state representation $\boldsymbol{z}_{\text{aff}}^s = z_{\text{aff}}^*(s)$, where any states s_i and s_j are mapped to the same value only when $f^*(s_i) = f^*(s_j)$. With a representation that could cluster states in this way, we could trivially determine the falsity of a collected negative $s \in \mathcal{D}_g^-$ by checking if $z_{\text{aff}}^*(s) = z_{\text{aff}}^*(s_j)$ for any $s_j \in \mathcal{D}_g^+$, that is, if we have encountered a true positive with the same representation. The classification procedure could be interpreted as "labeling" these contexts with affordance values. This is somewhat of a "chicken and egg" problem, since to learn affordances, we require a representation that maps states to contexts with the same affordance values, which clearly requires some prior knowledge about affordances. Fortunately, from earlier, we know that affordances will only change when either (1) a milestone is collected and (2) when edge stochasticity occurs. Since (2) is by definition a rare occurrence, states s_t and s_{t+1} are more likely than not to satisfy $f^*(s_t) = f^*(s_{t+1})$, so long as they exist in the same *segment* between milestones. In this case, we can say that s_t and s_{t+1} share the same *achievement context*, $\boldsymbol{z}^s = z(s)$. Let $z_\psi(s)$ be an achievement context embedding represented by a differentiable function parameterized by ψ. We can train $z_\psi(s)$ from experience using the following contrastive loss:

$$\mathcal{L}_\psi = \sum\nolimits_j \left[\left\| z_\psi(s_j^a) - z_\psi(s_j^p) \right\|_2^2 - \left\| z_\psi(s_j^a) - z_\psi(s_j^n) \right\|_2^2 + \alpha \right]_+ ,$$

where each s_j^a is a randomly chosen *anchor*, each s_j^p is a *positive*[5] example chosen within the segment according to a (truncated) normal distribution, $\mathcal{N}_T(0, \sigma^2)$, centered around (and excluding) s_j^a, s_j^n is chosen randomly among other segments and is treated as a *negative* example, and α is an arbitrary margin value. This loss pushes representations of states from the same achievement context together, and pulls representations of states from different contexts apart. We show in our

[5]Here, the usage of "positive" and "negative" refers to whether points share the same achievement context.

results that a wide range of σ produce useful representations. For our edge stochasticity experiments (Fig. 103) we use a low $\sigma = 2.0$ to reduce the risk of sampling across affordance changes.

False negative filtering In the learned representation space, we expect false negative points to be closer to positive points than true negatives. Given a negatively-labeled state s_q for classifier head f_g, we compute[6] the mean distance from $z_\psi(s_q)$ to the representations of the k closest positive points in a population uniformly sampled[7] from $\mathcal{D}_g^+$, denoted d_q^k. We expect d_q^k to be large for true negatives and small for false ones, and we can determine an effective separating margin in the following way. First we compute distance scores for a random sample of positive points, denoted $\{d_p^k\}$, to use as reference. We ensure these points come from segments that are disjoint from the population points' segments to avoid trivially low scores that might skew the distribution. We then fit a Gaussian distribution to $\{d_p^k\}$ and compute an upper confidence bound ρ for a given percentile value and confidence level. Any $d_q^k < \rho$ is very similar to positive points in the representation space, so we count s_q as a false negative and exclude it from our training set. Note, we do not train head f_g (and therefore do not reliably prune) until we have access to both positives and negatives for g.

Method overview The HAL architecture consists of a bi-level policy like h-DQN, a context embedding network, and an affordance classifier (see Fig. 31), which are all learned concurrently. Intuitively, the affordance classifier is able to generalize to a novel state, s, by first identifying the abstract achievement context, $z_\psi(s)$, associated with the state, and then outputting an affordance value based on previous experience in that context. If $z_\psi(s)$ has also not been encountered, that context will not be strongly "labeled" either way, so we will not be invariably pruning it. The meta-controller selects a subtask g at the beginning of the episode, and selects a new subtask g' after collecting *any* milestone or whenever an option times out after a predefined number of steps (our β_g). At each step, the current state is fed to the controller, which outputs an action conditioned on the most recently selected subtask. After discretizing the classifier's output to a binary mask, we perform an affordance aware version of ϵ-greedy as follows. Given parameters ϵ_{aff} and ϵ_{mc}, we select a random subtask within the mask with probability ϵ_{aff}, randomly across all subtasks with probability ϵ_{mc}, and otherwise select greedily with respect to meta-Q *within* the mask.

[6] This procedure is akin to the particle entropy approach used in [168].

[7] For efficiency, we sample just enough points so that we are likely to cover all encountered contexts.

4 RESULTS AND DISCUSSION

This section presents the evaluation procedures, results, and discussion for both our integrated lifelong learning framework (Sections 4.2–4.3.3), the methods and approaches that make up its individual modules (Section 4.4), and additional lifelong learning research conducted over the course of the program (Section 4.5). We begin by establishing procedures common to the integrated framework evaluation.

4.1 Common Evaluation Procedures and Metrics

The integrated system experiments of Sections 4.2 and 4.3 were conducted over lifetimes defined by a common set of task sequencing scenarios, and evaluated by a common set of lifelong learning metrics, both of which were defined as part of the L2M program. The scenarios consisted of alternating learning blocks for individual tasks and evaluation blocks over all tasks, with task selection, task ordering, and learning block lengths defined by the following scenario types:

- **Condensed.** Lifetimes include the full set of tasks. Each task occurs for only a single learning block. Learning block lengths are set as the approximate length required for a single task learner to reach saturation over training performance.
- **Dispersed.** Lifetimes include the full set of tasks. Each task occurs for three learning blocks, where tasks are repeated only after every other task has had a learning block. Learning block lengths are set as one third of the approximate length required for a single task learner to reach saturation over training performance.
- **Permuted.** Lifetimes include the full set of tasks. Each task occurs for two learning block, where tasks are repeated only after every other task has had a learning block. Learning block lengths are set as half of the approximate length required for a single task learner to reach saturation over training performance.
- **Alternating.** Lifetimes include two tasks selected by random sampling. The tasks are trained in alternating learning blocks until three learning blocks occur per task. Learning block lengths are set as one third of the approximate length required for a single task learner to reach saturation over training performance.

The common set of lifelong learning metrics are defined in detail by New et al. [169], but we provide a brief summary of what each metric represents here.

- **Performance Maintenance (PM).** Resistance to catastrophic forgetting; values of > 0 shows no forgetting
- **Forward Transfer (FT).** Jumpstart, or how training previous tasks changes the initial training performance for subsequent tasks; values of > 1 show beneficial jumpstart
- **Backward Transfer (BT).** Change in evaluation performance for a given task that occurs after training a different task; values of > 1 show beneficial transfer

- **Relative Performance (RP).** Ratio of lifelong learning agent performance compared to the performance of a Single-Task Expert (STE); values of > 1 show a lifelong learning performance gain
- **Sample Efficiency (SE).** Ratio of lifelong learning agent time to reach training performance saturation compared to STE training time; values of > 1 show a lifelong learning efficiency gain

4.2 APL MiniGrid Benchmark

To evaluate the core of our integrated system's RL pipeline in a setting that supports longer task sequences and faster experimentation, we applied our L2M framework with the core learning algorithm of LPG-FTW to APL's MiniGrid benchmark setting, which was developed from default tasks in the MiniGrid environment [170]. All experiments involved our agent learning six tasks with three variants each, randomly ordered for both Condensed and Dispersed scenarios. The tasks, shown in Figure 32, are as follows:

- *Simple Crossing*: navigate a simple maze to reach a goal; variants change grid size and number of crossings
- *Distributional Shift*: navigate to goal while avoiding lava; variants change lava strip position
- *Dynamic Obstacles*: navigate to a goal while avoiding moving obstacles; variants change grid size and number of obstacles
- *Custom Fetch*: pick up a specific object; variants change grid size and number of objects
- *Custom Unlock*: pick up a key and unlock a door; variants change grid size
- *Door and Key*: navigate to a goal after unlocking a door with a key; variants change grid size

4.2.1 Condensed and Dispersed Scenario Results.

Both the Condensed and Dispersed scenarios consisted of running 100 lifetimes, each with a different randomized task ordering. We compared our lifelong learning agent to single-task experts (STEs) trained for each task variant, with 10 STEs trained per task variant. The aggregate metrics for each scenario, shown as mean $\pm$ standard deviation, are given in Table 3.

Table 3: Aggregate Measures for Condensed and Dispersed MiniGrid Scenario Results

Scenario	PM (> 0)	FT (> 1)	BT (> 1)	RP (> 1)	SE (> 1)
Condensed	-15.32 $\pm$ 5.57	4.18 $\pm$ 0.94	1.12 $\pm$ 0.18	1.04 $\pm$ 0.06	1.32 $\pm$ 0.58
Dispersed	-4.76 $\pm$ 2.33	3.55 $\pm$ 0.82	1.04 $\pm$ 0.04	1.03 $\pm$ 0.06	1.15 $\pm$ 0.26

The benefit of our LPG-FTW approach is that it speeds up learning on subsequent tasks as compared to single-task learners. This is shown by high forward transfer and sample efficiency measures,

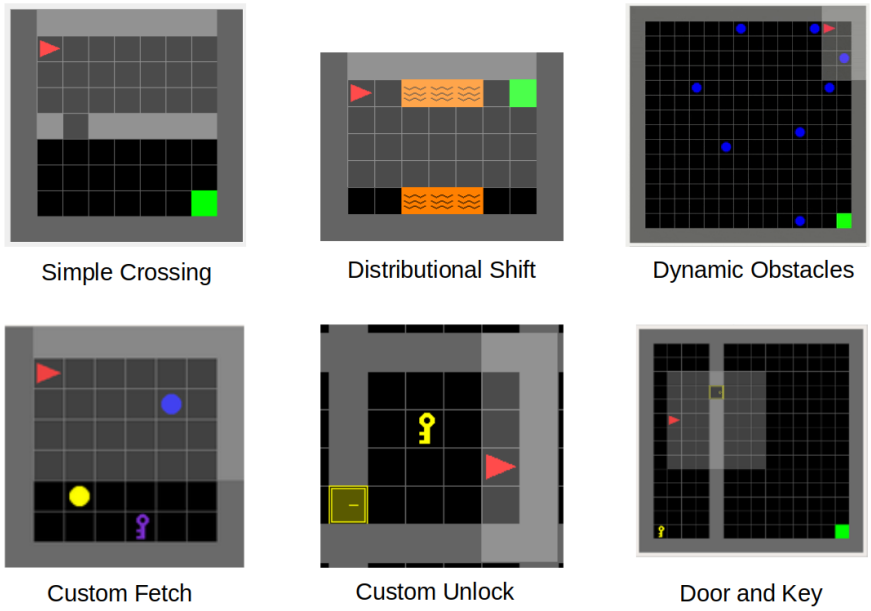


Figure 32: Example Task Visualizations for the APL MiniGrid Benchmark

which indicate that the lifelong learning agent effectively leverages information learned from prior tasks to both start learning from a higher point and reach the maximum performance faster than STEs when presented with novel tasks. Further, the factorized nature of the algorithm gives some protection against catastrophic forgetting, as evidenced by the greater than one backward transfer, and the relatively close to zero performance maintenance (for context, training a single naïve PPO model over the same curricula gives an average performance maintenance of approximately -15). This performance can be seen in the training and evaluation curve plots of selected representative lifetimes for the Condensed scenario in Figure 33, and for the Dispersed scenario in Figures 34 and 35, shown and discussed below.

The main advantages of our LPG-FTW pipeline over STE learners can best be seen in the training curve comparisons to the STEs. Beneficial examples of forward transfer are best shown by large jumps in performance at the start of a learning block. Many tasks learned later in the agents' lifetimes start training at significantly higher rewards, which is most apparent in the Condensed scenario such as `doorkey_s7`, `customunlock_s5`, and the later `simple_crossing` and `dynobstacles` tasks in Figure 33 (bottom). While the magnitude of the performance increase is lower, the same pattern appears in many tasks in the Dispersed scenario, such as `doorkey_s7`, and the later `distshift`, `dynobstacles`, and `simplecrossing` tasks in Figure 34 (bottom), and the later `doorkey`, `customunlock`, and `simplecrossing` tasks in Figure 35 (bottom). Improvements in sample efficiency can be seen by training curves that dominate the STE training curves, examples of which are also apparent in both scenarios, such as `simplecrossing_s9n3`, `simplecrossing_s9n1`, `distshift_r3`, and `dynobstacles_s10n3` in Figure 33, `dynobstacles_s6n1` and `dynobstacles_s8n2` in Figure 34, and `simplecrossing_s9n1` and `customunlock_s7` in Figure 35. We also note that for most tasks, our approach closely tracks the STE training curves. As such, for a large majority of tasks in this MiniGrid domain, LPG-FTW at worst gives comparable sample efficiency to a single-task learner, while giving measurable

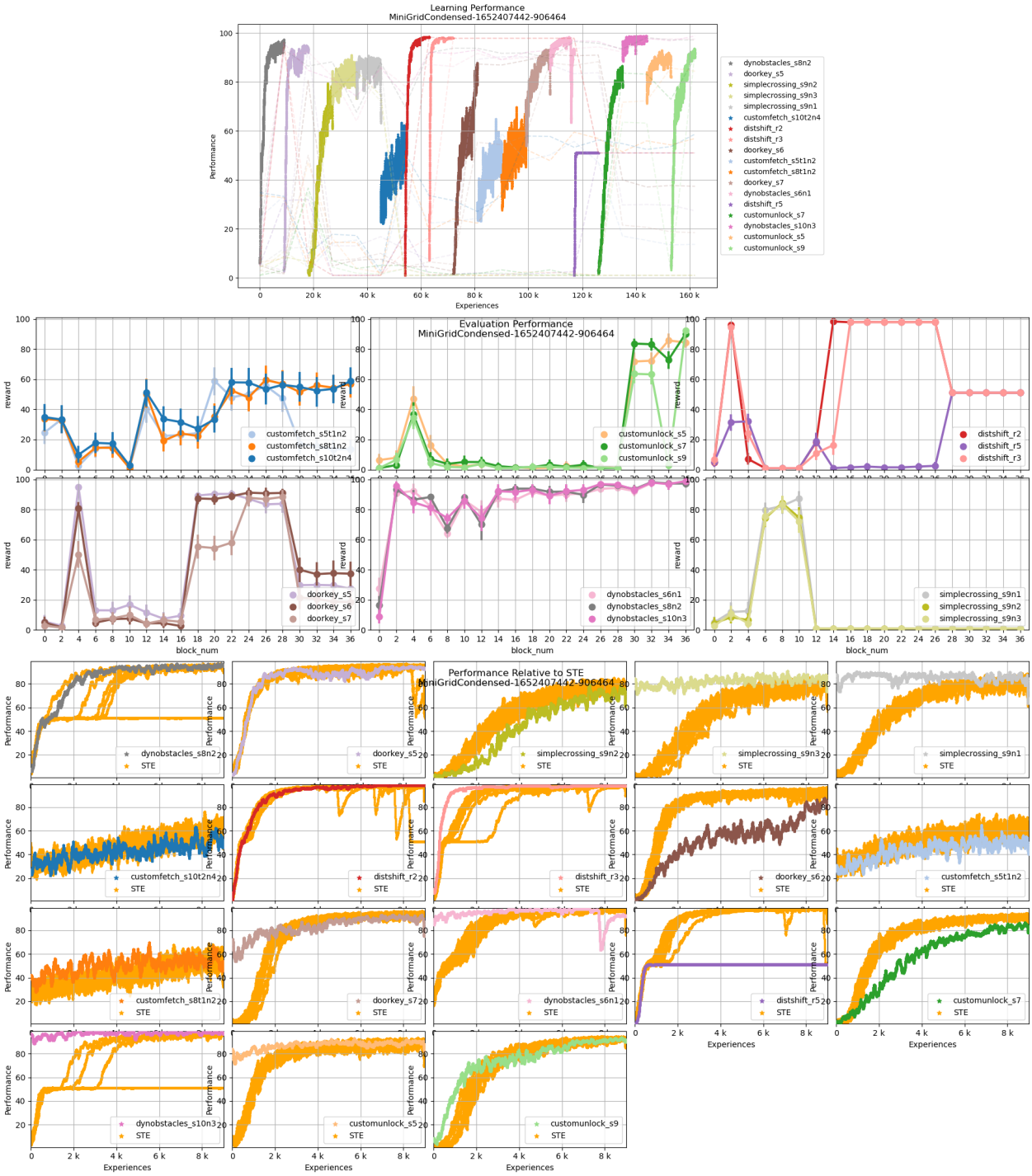


Figure 33: Training (top), Evaluation (middle), and STE Comparison (bottom) for Condensed MiniGrid Scenario

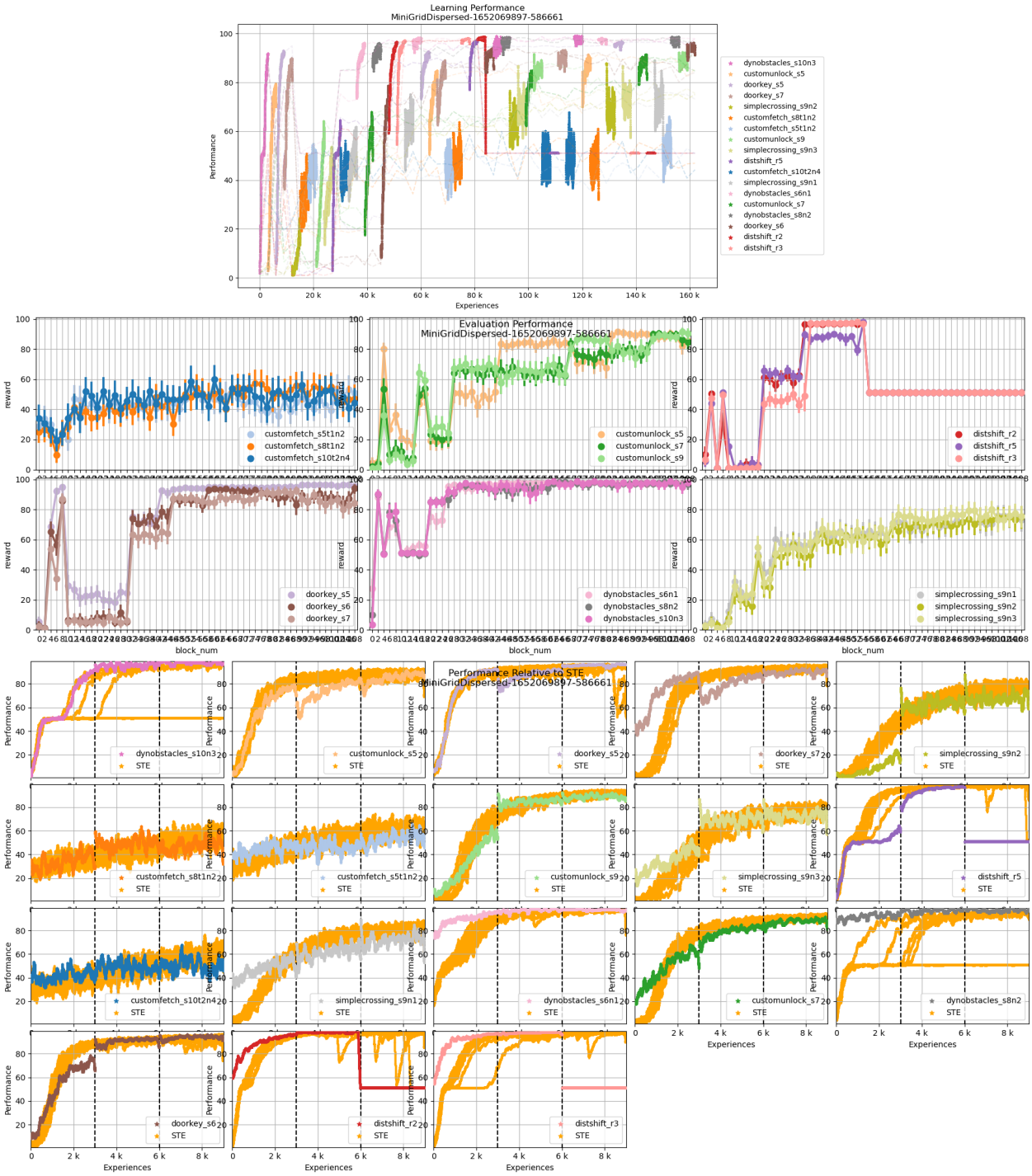


Figure 34: Training (top), Evaluation (middle), and STE Comparison (bottom) for Dispersed MiniGrid Scenario, First Example

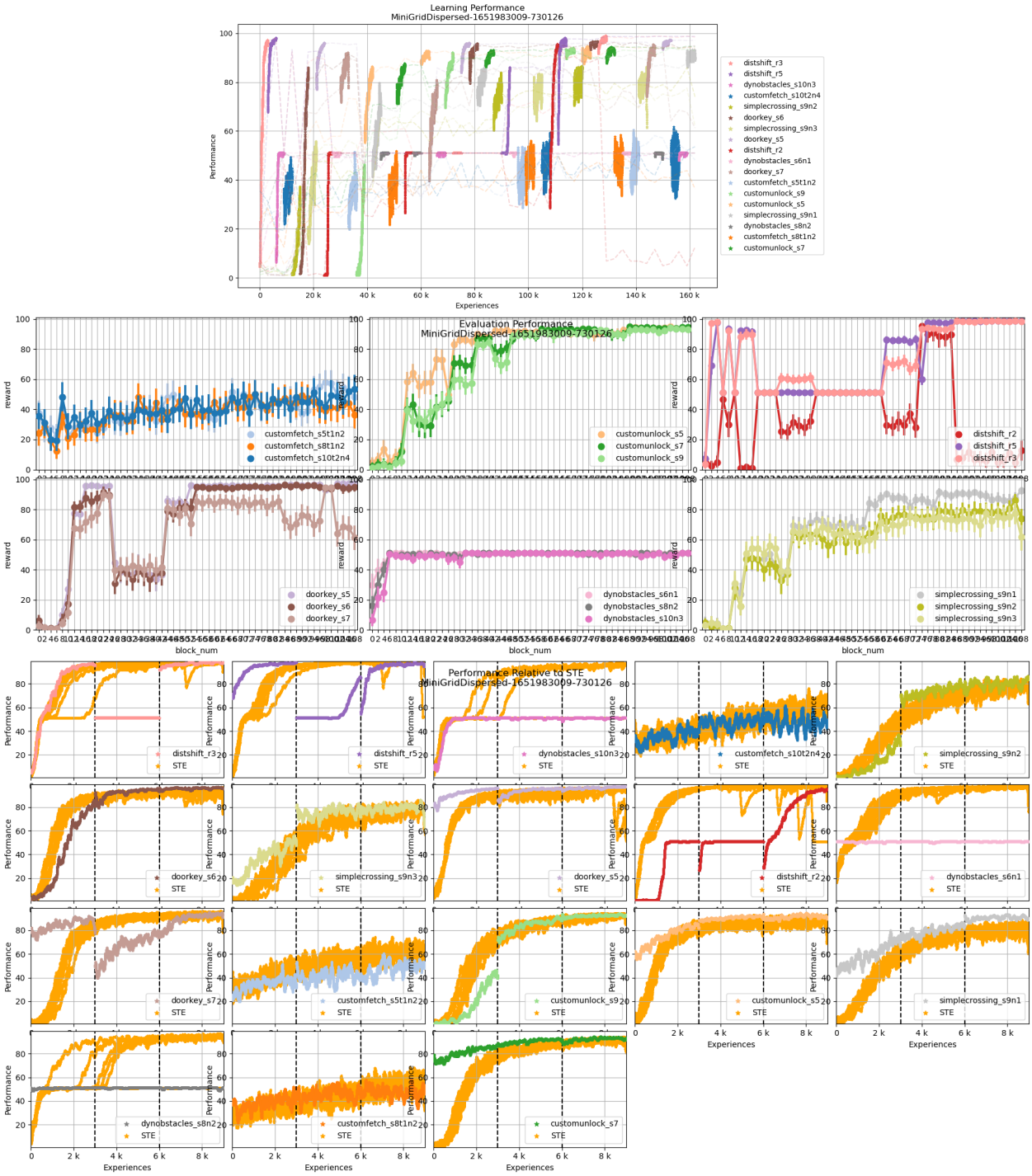


Figure 35: Training (top), Evaluation (middle), and STE Comparison (bottom) for Dispersed MiniGrid Scenario, Second Example

improvements for some tasks.

The protection against catastrophic forgetting afforded by the factorized nature of LPG-FTW can best be seen by the evaluation curves, which are computed for the full set of tasks after each individual learning block. While we do see some cross-task interference, typically in the Condensed scenario and in the early stages of the Dispersed scenario, most tasks typically maintain steady performance from one evaluation block to the next. There are also a few clear cases of beneficial backward transfer, such as the `dynobstacles` tasks in Figure 33 (middle) and the `simplecrossing` tasks in Figure 34 (middle), which all show steady evaluation performance increases across the agent's full lifetime.

Our final observation from inspecting individual runs is that the task order does seem to affect lifelong learning performance. This can be seen by comparing the `dynobstacles` and `distshift` tasks between our two presented Dispersed scenario lifetimes. The dynamic obstacles and distributional shift tasks are particularly difficult, as they are the only tasks that include failure conditions that terminate a run and give the agent negative reward (hitting an obstacle or hitting lava, respectively). When one or more of these tasks are included in the first k tasks in the sequence, they are used to train one of the agent's k factors, which are used to generate policies for subsequent tasks. For these experiments, we set $k = 4$ to ensure that some of the task types would not be used to learn the initial factors. Further, when learning the first k tasks, the agent initializes its policy for a new task using a previous task policy, and thus the first task in the sequence has a large influence on what and how agents learn tasks in the future. The task sequence in Figure 34 starts with a `dynobstacles` task, but has no `distshift` tasks within the first four tasks. As a result, the lifelong learning agent effectively learns all of the `dynobstacles` tasks by the end of its lifetime, while all of the `distshift` tasks have significant performance drops due to interference from other tasks. Conversely, the task sequence in Figure 35 starts with two `distshift` tasks, and the agent is able to learn `distshift` tasks more effectively throughout its lifetime, even recovering from some of the similar task interference seen in the previous Dispersed scenario example. Also of note is that while there is a `dynobstacles` task within the first four tasks, it follows the two `distshift` tasks, and we see that this agent fails to learn the `dynobstacles` task, and further cannot learn any `dynobstacles` tasks over the rest of its lifetime. These observations on task ordering effects suggest that the agent's curriculum is important for our LPG-FTW pipeline, motivating work on the curriculum learning component.

4.2.2 Evaluation of Learning Block Lengths.

We conducted an additional experiment to test the hypothesis that our LPG-FTW approach would perform better for all lifelong learning metrics if given longer learning blocks in the dispersed scenario. The reasoning behind this is that since LPG-FTW is a factorized method, performance on subsequent tasks is dependent on how well the agent learns the tasks during the initial k learning blocks, as these k policies form the base factors that are combined to learn future novel tasks. Since the Dispersed scenario defines learning blocks as one third the length required for an STE to learn the task, we were concerned that the LPG-FTW agent's k factors, consisting of only partially-learned tasks, would negatively impact the system's ability to learn in a lifelong setting. To test this hypothesis, we ran two conditions of the Dispersed scenario each over 10 different task orderings. The first condition, which we denote *ShortLB*, used the same learning block length of the

official scenario. The second condition, *LongLB*, tripled the learning block length. In the case of our system, this resulted in learning block lengths of 3000 episodes vs. 9000 episodes.

We summarize differences between the lifelong learning performance using the program metrics in Table 4, and provide a side-by-side comparison of the task learning curves for a single lifetime over the same task ordering in Figure 36. From this experiment we actually did not find support for our hypothesis, and in fact see better overall performance when using shorter learning blocks, although this claim has some nuance. The shorter learning block lengths result in less catastrophic forgetting, as shown by higher performance maintenance, and higher relative performance compared to the STEs. On the other hand, longer learning block lengths amplify the faster training benefits of LPG-FTW, showing higher forward transfer and sample efficiency, but achieve lower STE relative performance. Interestingly, we see the exact same trade-offs between PM, FT, and SE when comparing the Condensed and Dispersed scenario metrics from the experiments of Section 4.2.1 (see Table 3), where the Condensed scenario has significantly longer learning blocks than the Dispersed scenario.

We conjecture that with longer learning blocks, the agent learns highly effective policies that overfit to certain types of tasks; the initially learned policies achieve high reward for many tasks, but are explicitly unable to learn some tasks types. With shorter learning blocks, the agent learns incomplete policies that afford more flexibility, which despite lower peak performance can still learn the full set of tasks. This behavior is clearly exemplified by comparing the learning curves over the full set of tasks between any two lifetimes that use the same curriculum seed, as shown in Figure 36. The LongLB agent experiences irrecoverable performance drops for many of the `doorkey` and `distshift` tasks, while being unable to fully learn the two later `distshift` tasks, whereas the ShortLB agent shows consistent learning and ability to recover from performance drops over the full task set. Additionally, as the only difference between lifetimes within each condition are task ordering, we observe that using shorter learning blocks provides more robustness to changes in task order, supported by the lower standard deviations across all of the lifelong learning metrics in Table 4. We conclude that, for LPG-FTW, the flexibility provided by incomplete policies learned over relatively short learning block lengths are preferable to the highly-specialized policies learned over long learning block lengths.

Table 4: Aggregate Measures for Dispersed Scenario Learning Block Length Experiment

Condition	PM (> 0)	FT (> 1)	BT (> 1)	RP (> 1)	SE (> 1)
ShortLB	**-3.61 $\pm$ 1.36**	3.57 $\pm$ 0.85	1.06 $\pm$ 0.06	**1.04 $\pm$ 0.06**	1.19 $\pm$ 0.36
LongLB	-9.60 $\pm$ 1.46	**4.23 $\pm$ 0.95**	1.14 $\pm$ 0.18	0.91 $\pm$ 0.06	**1.44 $\pm$ 0.77**

4.3 Lifelong Robot Learning for the Visual Scavenger Hunt

Our primary evaluation setting which drove development of our integrated lifelong learning framework is the service robot visual scavenger hunt domain detailed in Section 3.2. We evaluate our integrated lifelong learning framework, shown in Figure 1. Depending on its configuration, the integrated system is capable of lifelong learning of interleaved classification/regression and RL tasks, by passing the task to the pipeline corresponding to the appropriate learning paradigm. We present a progressive series of evaluations focusing on different configurations for image classification tasks in the visual scavenger hunt domain, followed by object search RL tasks in the same

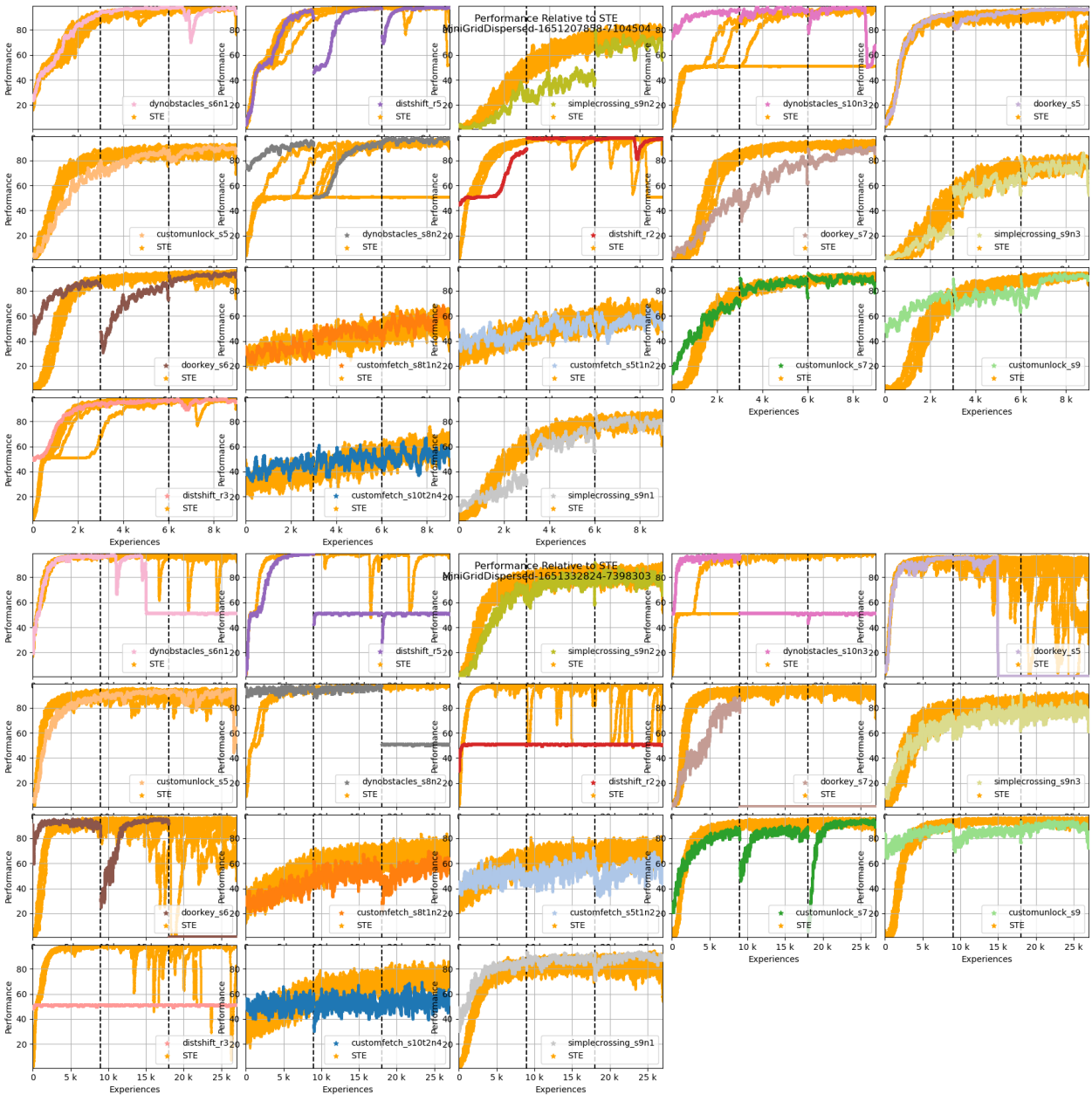


Figure 36: Comparison of Training Curves for Dispersed MiniGrid Scenario ShortLB (top) and LongLB (bottom) Conditions, Same Curriculum Seed

domain. Additionally, we evaluate a final configuration of our framework that tightly integrates the classification and RL pipelines. We show promising results for each single-paradigm pipeline, suggesting that further research in the fully-integrated setting may prove effective for deployed service robots in the future, but that the tightly-integrated classification and RL lifelong learning problem we have identified is significantly more complex than traditional lifelong learning settings, and will thus require significant advancements in the field of lifelong learning to reach sufficient performance for deployment on real service robots. In light of our results, we conclude with a discussion of the challenges of developing a multi-learning-paradigm system for a realistic service robot task, and identify future directions of open research in this area.

4.3.1 Classification Experiments.

We evaluated our perception pipeline in using the visual scavenger hunt classification setting described in Section 3.2. The perception pipeline itself consisted of the following blocks from the full system depicted in Figure 1: the environment, controller, and either the DF-CNN or META-KFO classification models, with the optional addition of the META-KFO meta-learner module. Our final best-performing classification system used the DF-CNN instead of the META-KFO classification model. We present results from the DF-CNN version of the pipeline below, followed by results from an experiment that directly compared the DF-CNN and META-KFO pipelines.

We evaluated the DF-CNN pipeline over lifelong learning curricula composed of 4-way classification tasks for three superclasses, instantiated over two distinct house layouts (6 tasks in total, 3 tasks $\times$ 2 environments). We randomly sampled different task orderings according to the Condensed and Dispersed scenarios discussed previously. We present aggregate results (mean $\pm$ standard deviation) over all of the experiment lifetimes in Table 5, and provide a representative learning curve for both the condensed and dispersed scenarios in Figure 37, where light grey backgrounds indicate learning blocks, and dark gray backgrounds indicate evaluation blocks. This experiment was part of the official program evaluation, and detailed scenario specifications, system description, and raw results can be found in the `m15_eval` directory of the `darpa-l2m/sg_upenn_eval` repository.

Table 5: Aggregate Measures for Perception Pipeline in Visual Scavenger Hunt Domain

Configuration	PM (> 0)	FT (> 1)	BT (> 1)	RP (> 1)	SE (> 1)
DF-CNN (M15)	0.01 ± 0.30	1.00 ± 0.01	1.00 ± 0.00	2.12 ± 0.08	1.71 ± 0.27

The main advantage of the DF-CNN pipeline is that it significantly speeds up task learning compared to single task learners, as shown by the high RP and SE measures and the dominant learning curves of Figure 37 (right). Since the DF-CNN is a factorized method, it also provides effective mitigation of catastrophic forgetting, as shown by the PM and BT values of approximately 0 and 1, respectively. This property can be seen in the stable performance of the learning curves in Figure 37 (left). Once a task is learned, its evaluation performance does not drop after other tasks are learned. Further, in the dispersed scenario, the each training curves for tasks in each subsequent learning block picks up almost exactly where it left off during the corresponding previous learning block.

We conducted an experiment to directly compare the DF-CNN perception pipeline with the META-KFO pipeline. For this experiment, lifelong learning curricula consisted of a randomly sampled ordering of 4-way classification tasks for four superclasses, repeated two times each. We present a

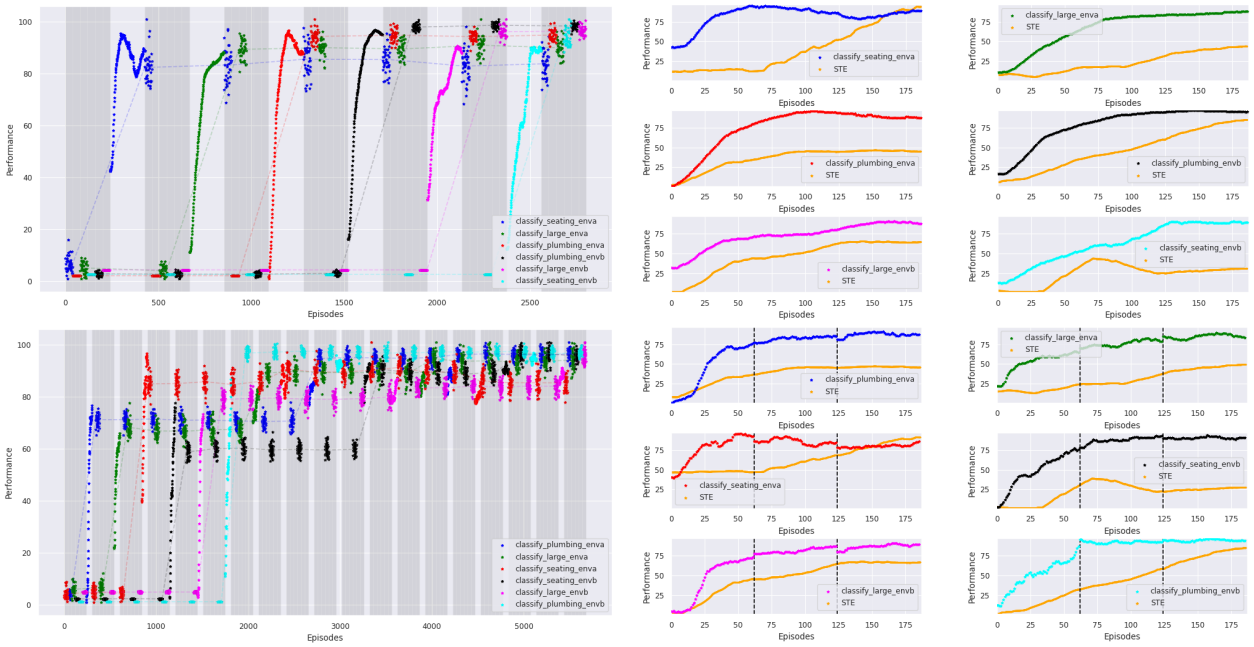


Figure 37: Sample Learning Curves and STE Comparisons for One Lifetime of the DF-CNN Perception Pipeline, Condensed (top) and Dispersed (bottom) scenarios

comparison of the aggregate results (mean $\pm$ standard deviation) for the program-defined metrics over both conditions of the experiment lifetimes in Table 6, and provide a visual comparison of each pipeline's behavior with side-by-side learning curves in Figure 38. This experiment was part of the official program evaluation, and detailed scenario specifications, system description, and raw results can be found in the `m12_eval` and `m12_experiments` directories of the `darpa-l2m/sg_upenn_eval` repository.

Table 6: Aggregate Measures for Alternative Perception Pipelines in Visual Scavenger Hunt Domain

Configuration	PM (> 0)	FT (> 1)	BT (> 1)	RP (> 1)	SE (> 1)
DF-CNN (M12)	**-0.44 $\pm$ 1.12**	1.01 $\pm$ 0.09	0.99 $\pm$ 0.02	1.94 $\pm$ 0.26	1.61 $\pm$ 0.12
META-KFO (M12)	-20.81 $\pm$ 15.22	1.00 $\pm$ 0.00	0.91 $\pm$ 0.07	**2.38 $\pm$ 0.40**	**3.40 $\pm$ 0.46**

The goal of this experiment was to characterize any differences in performance between factorized classification models, as represented by the DF-CNN system configuration, and meta-learned classification models, as represented by the META-KFO system configuration, in a lifelong supervised learning setting. The results show that, while both approaches show good lifelong learning performance, META-KFO provides faster learning (higher SE) whereas the DF-CNN provides more stable learning by better catastrophic forgetting mitigation (higher PM and BT, with lower standard deviations). All of this is illustrated in the sample learning curves of Figure 38. The difference in learning speed is shown by the DF-CNN taking multiple learning blocks to fully learn the task compared to META-KFO, which saturates training performance for each task in a single learning block. The difference in stability of learned tasks can be seen by the DF-CNN's stable evaluation block performance, whereas the META-KFO pipeline shows drops in evaluation performance as

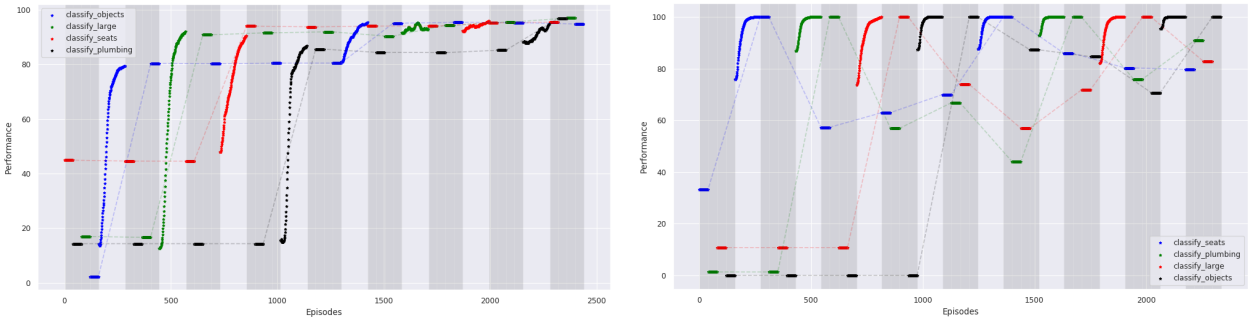


Figure 38: Sample Learning Curve Comparison for One Lifetime of the DF-CNN (left) vs. META-KFO (right) Perception Pipelines

soon as a different task is trained. We conclude that the DF-CNN's mitigation of catastrophic forgetting is more desirable than the META-KFO pipeline's faster but less stable task learning, as once a task is learned, we can essentially guarantee reliable performance at any point in the future, and as such we prioritized the use of the DF-CNN perception pipeline configuration in our integrated pipeline experiments.

4.3.2 Reinforcement Learning Experiments.

We evaluated our RL pipeline in using the visual scavenger hunt reinforcement learning setting described in Section 3.2. In particular, we present the results of a series of experiments using different RL system configurations developed and tested over the course of the L2M program, the results of which act as ablation studies over subsets of the components shown in Figure 1. The system configurations we evaluated are:

- **M12** Intrinsic-motivation-focused lifelong learning agent. Modules include the intrinsic motivation module, A2C RL models, and the controller and environment modules.
- **M15** Factorized lifelong learning agent. Modules include the LPG-FTW RL model, and the controller and environment modules.
- **M18** Optimized factorized lifelong learning agent. The agent used the same limited set of modules as the M15 agent, but the system itself included additional tuning based on the results of the M15 experiments.

Experiments for each pipeline consisted of alternating training and evaluation blocks for the lifelong learning agent over a randomly ordered sequence of tasks, with learning block lengths determined by a given experimental scenario. The M12 experiments used the four RL tasks from the reinforcement learning setting described in Section 3.2, evaluated over the Permuted and Alternating scenarios. The M15 and M18 experiments used the Condensed and Dispersed scenarios over three of the RL tasks instantiated in two distinct house layouts (6 tasks in total, 3 tasks $\times$ 2 environments). These experiments were part of the official program evaluation, and detailed scenario specifications, system descriptions, and raw results can be found in the corresponding `m<n>_eval` directories of the `darpa-l2m/sg_upenn_eval` repository.

A summary of results for each pipeline over the program-defined performance metrics is given in Table 7, and specific characteristics of the lifelong learners are illustrated for each approach in learning curves for individual sampled lifetimes in Figure 39, where light grey cells indicate learning blocks and dark grey cells indicate evaluation blocks. Each of these experiments provide some insight into both the mechanisms behind lifelong learning and challenges in lifelong learning system development, which we discuss below.

Table 7: Aggregate Measures for Alternative RL Pipelines in Visual Scavenger Hunt Domain

Metric	M12	M15	M18
PM (> 0)	-60.14 $\pm$ 21.52	-13.95 $\pm$ 20.49	**4.37** $\pm$ **11.25**
FT (> 1)	0.89 $\pm$ 0.80	1.95 $\pm$ 0.97	**3.11** $\pm$ **2.36**
BT (> 1)	1.20 $\pm$ 1.56	1.19 $\pm$ 0.16	1.11 $\pm$ 0.07
RP (> 1)	0.75 $\pm$ 0.07	0.75 $\pm$ 0.06	**0.88** $\pm$ **0.03**
SE (> 1)	0.66 $\pm$ 0.27	1.88 $\pm$ 1.96	0.83 $\pm$ 0.03

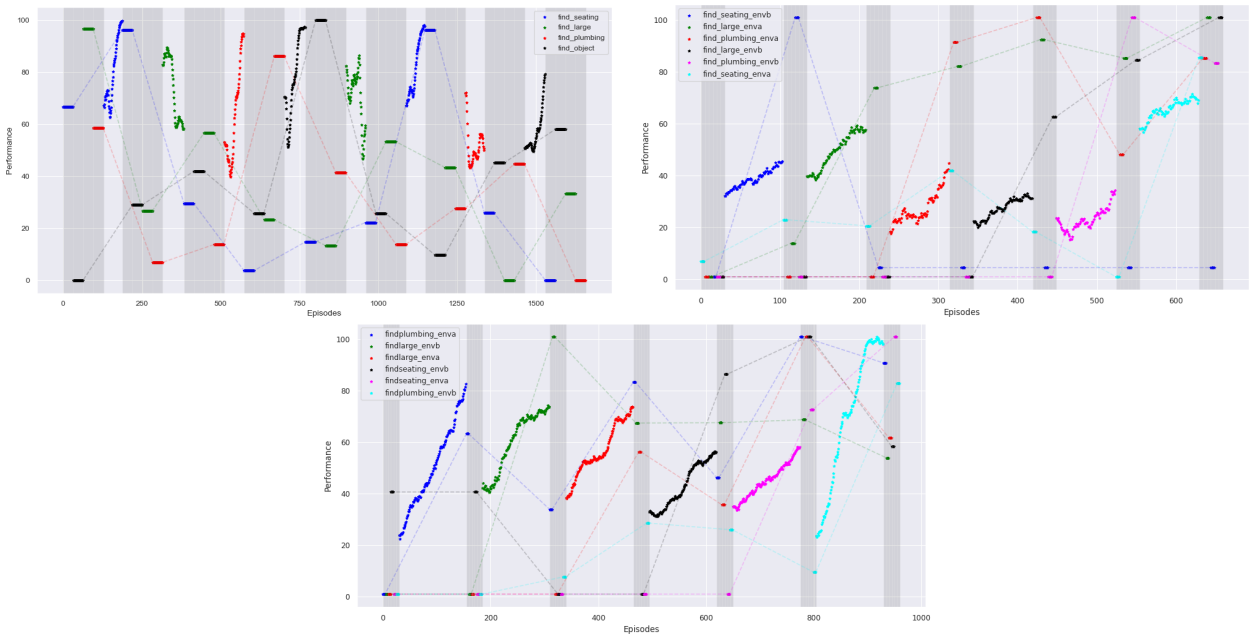


Figure 39: Sample Learning Curves for One Lifetime of the M12 (left), M15 (right), and M18 (bottom) RL Pipelines

Our first RL experiment (M12) hypothesized that intrinsic motivation would improve FT, RP, and SE in a lifelong learning settings, making it an effective mechanism for knowledge reuse in lifelong RL. Our results did not support our hypothesis, instead showing that intrinsic motivation is not an effective mechanism for lifelong learning, as shown by the low measures across all metrics in the M12 column of Table 7. The main issue we identified was that the system was highly susceptible to catastrophic forgetting, as evidenced by the particularly low PM score. We see some instances of backward transfer, where training on one task improves the evaluation performance of other tasks, but overall the evaluation performance is unstable due to cross task interference. This characteristic

is clearly shown by the shifting evaluation block performance in Figure 39 (left), and by the fact that the second set of training blocks for each task start at significantly lower reward values than those reached at the end of the initial training blocks. We conclude that intrinsic motivation alone is not an effective mechanism for lifelong learning, and would likely need to be combined with approaches specifically designed to mitigate catastrophic forgetting, such as factorized methods. Such an integration presents a non-trivial research problem which could be explored in future work.

Our next set of RL experiments (M15) focused on evaluating the effectiveness of our factorized LPG-FTW algorithm in the highly complex and realistic Habitat/Matterport environment. The results show significant improvement compared to our intrinsic motivation pipeline across all of our lifelong learning metrics, with the exception of comparable relative performance. We note that while the PM score was still negative, it is significantly higher than the intrinsic motivation pipeline, which shows increased mitigation of catastrophic forgetting. This effect can be seen in the comparatively more stable evaluation block performance of Figure 39 (right), particularly after the first three tasks are trained, which instantiates the full set of factors used to learn subsequent task policies. One issue we observed, however, is that the later tasks were reaching training saturation very quickly, as shown by the last three tasks in Figure 39 (right). We hypothesized this issue arose because we had to limit the agent's training time due to the high computational time and resource requirements of training agents in the habitat environment. As such, the initial three tasks were not fully learned, and thus the factors used to learn the final three tasks consisted of immature policies.

We focused specifically on addressing this issue for M18, rather than developing and testing a new system configuration. We continued to develop the LPG-FTW-based system configuration with additional network architecture search and hyperparameter tuning, specifically targeting faster learning rates to enable experimentation with longer learning blocks. As shown in the M18 column of Table 7, this optimization had the positive effect we hypothesized, resulting in significant improvements to both PM and FT. We also see some consistent examples of beneficial backward transfer, for example after training the fifth (magenta) task in Figure 39 (bottom), we see an evaluation performance improvement in all other previously trained tasks. This experiment still identified some key challenges in lifelong learning system development for service robots. Contrary to the experimental results in our original LPG-FTW paper [25], and despite the beneficial effects of high PM, FT, and BT, we do still see relatively low performance with respect to comparable single task learners (i.e. in the RP and SE metrics). We hypothesize that this performance drop is due to the increased challenge of learning in the high fidelity environments, and the higher task complexity that such an environment entails. Additionally, while we were able to run experiments with longer learning block lengths, we still had to limit learning block lengths to end before training performance saturation, as each individual lifetime took 4-6 days to run. These challenges motivated pursuit of systems-level solutions to reduce the complexity of the inputs to the RL models by leveraging prior knowledge learned by the perception system earlier in a deployed agent's lifetime, which we explored in our integrated systems experiments. These are still challenging problems that rely on highly-complex systems, and as such we believe further research on lifelong learning systems in real-world environments is a fruitful direction for future work.

4.3.3 Integrated System Experiments.

As a culmination of integrated framework for lifelong learning evaluation, we performed experiments with a tightly-integrated classification and RL pipeline, with the goal of leveraging lifelong learning

classification knowledge to improve the performance and training efficiency of a lifelong learning RL agent. Such a system represents a novel research area of multi-paradigm lifelong learning, and as such we view experiments in this area as stretch goals for our framework.

With little prior work in this area, we initially explored the potential of using classification features to boost the performance of an RL agent in multiple RL environments. Our early results, which are currently being submitted for publication, suggest a fundamental mismatch between the features learned for classification tasks and the features required for RL tasks, which we discuss briefly here. RL policies were trained over four environments by either training networks from scratch (De Novo), learning a set of classifier features in the same environment and transferring them directly as inputs to the RL policy (Frozen), or finetuning the transferred classifier features (Finetuned), the results of which are shown in Figure 40. While there's no clear pattern across all environments, we observe that for the simpler environments (MSR Jump and DMC Walker-Run [171]) the frozen transferred classifier features actually hinder RL policy learning. Further, the features for the MSR Jump task were verified to perfectly solve the evaluation tasks using a simple regression model, so the issue is not in the sufficiency of the features themselves, but in some particular qualities of learned features that are useful for deep reinforcement learning policies. Contrarily, for the more complex AI Habitat environments [31], we see that the frozen classifier features actually perform better than fine-tuned features. We expected that a sufficient set of transferred, pre-trained classification features would only improve RL policy learning, but these initial results showed otherwise. We discuss the implications of this at the end of this section and in the Conclusion as potential for future work.

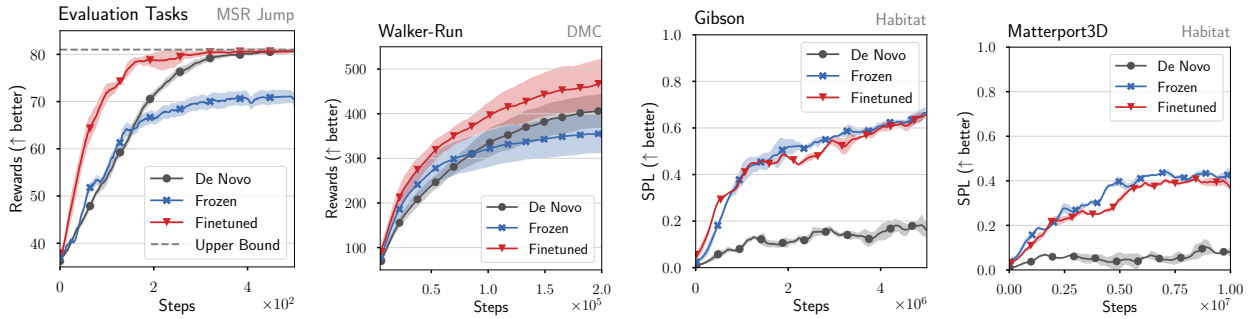


Figure 40: Early Experimental Results Snowing Mismatch between Transferred Classification and Learned RL Features

Based on these findings, we developed the multi-paradigm configuration of our integrated lifelong learning framework shown in Figure 41 (top). Rather than learning a shared set of features in an end-to-end manner, we instead leverage the results of a learned classification pipeline to adjust the agent's observation space into a lower-dimensional semantic map representation which contains a higher density of relevant information for object search RL tasks (i.e. occupancy probabilities and predicted object locations). The full pipeline works as follows. At each time step, an input image taken from the robot's rgbd sensor is passed into our active visuomotor exploration module (see Section 3.3.4), producing a top-down ego-centric occupancy grid. The agent uses the same input image to propose object candidates for classification, and label them using the DF-CNN classification pipeline (see Section 4.3.1), discarding any object proposals belonging to previously unseen classes. These two sources of information are combined to form an occupancy grid with

annotated object locations using the depth camera's measurements for each object labeled from the input image. We then use our learned object co-occurrence model (see Section 3.3.7) to expand the annotated map into a semantic map containing predicted object locations, with one channel per object type, which is then passed as the input into our LPG-FTW RL pipeline (see Section 4.3.2). Figure 41 (bottom) shows the training pipeline for the multi-paradigm configuration. During pre-deployment training, the system learns an object co-occurrence matrix and a vision-based exploration policy optimized for mapping, both of which are trained using standard (non-lifelong) algorithms in a held-out set of pre-deployment environments. During deployment, lifelong learning occurs in two phases: first, the object classifier is trained using the DF-CNN pipeline for all object recognition tasks excluding the targets of the object search RL tasks, and second, the system learns policies that solve the find object RL tasks leveraging the previously-learned distribution modeling, mapping exploration policy, and object classification model.

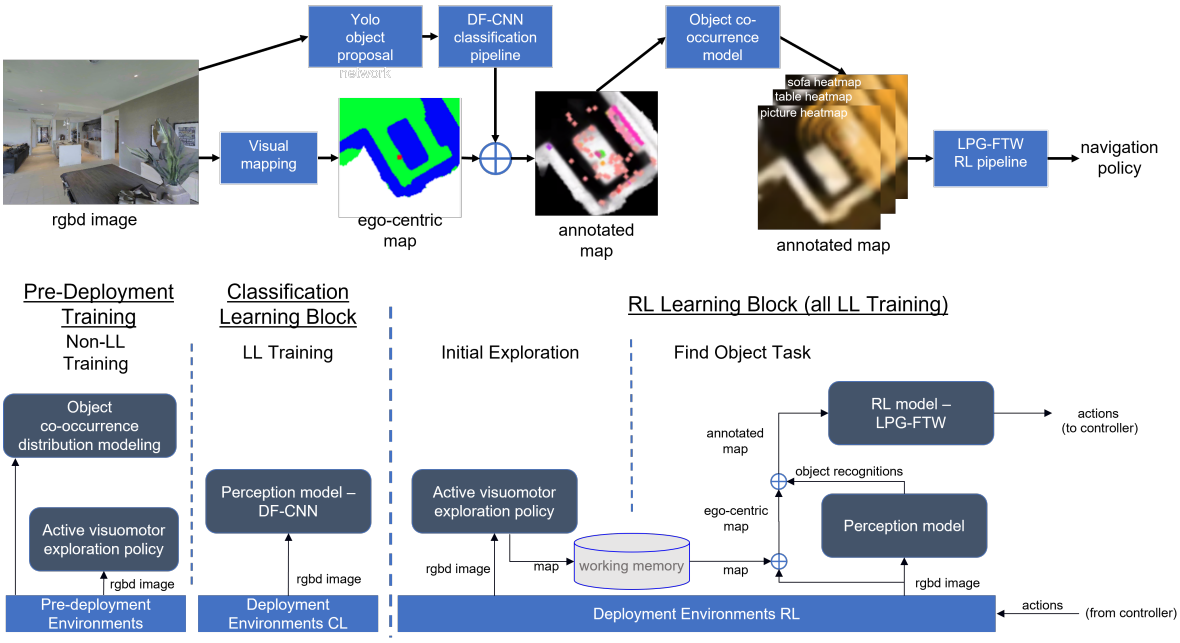


Figure 41: System Diagram for Tightly-Integrated Classification and RL Configuration (top) and Training Pipeline for Tightly-Integrated System (bottom)

After extensive architecture and hyperparameter search for the RL component of the multi-paradigm configuration, we found that output of the classification, mapping, and object co-occurrence models were not conducive to effective lifelong RL for our evaluation tasks. For completeness, we present results from selected hyperparameter configurations in Figure 42 (top). Our experiments consisted of lifetimes reduced to three RL tasks, as the multi-paradigm system took significantly longer to train than the configurations which used separate parallel pipelines. For comparison, the plots show results scaled relatively to the STE results produced for the M18 configuration of the reinforcement learning experiments presented in Section 4.3.2. The main difference between the M18 RL pipeline and the multi-paradigm configuration is that the multi-paradigm system leverages input from models that were previously learned to solve non-RL tasks. Leveraging this additional knowledge provided an immediate initial boost in performance, but as shown by the learning curves, the agent was then incapable of improving performance further. To verify this behavior more fully, we ran

corresponding experiments with learning blocks extended to four times the length of the M18 protocol, results of which are presented in Figure 42 (bottom).

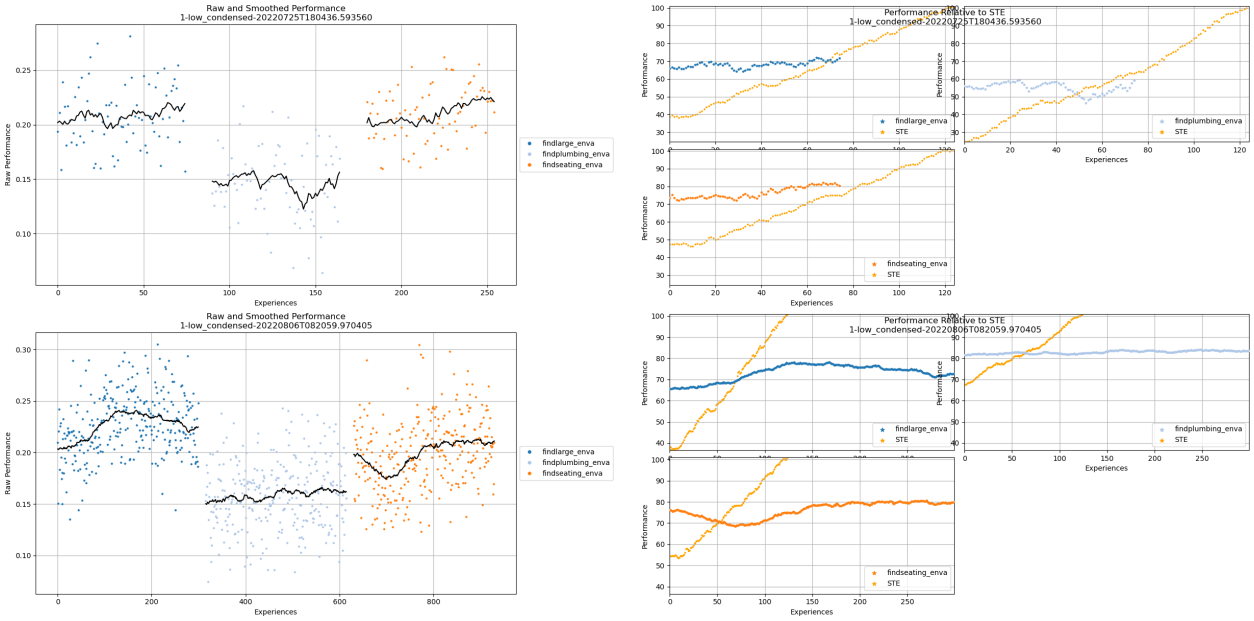


Figure 42: Selected Learning Curves for the Multi-Paradigm Integrated Lifelong Learning Framework Configuration

These results offer further support of the early experiments conducted by our USC group–a sufficient set of learned classification features can hinder RL learning performance when transferred to RL tasks–which seems to hold in the more complex service robot visual scavenger hunt domain. We hypothesize that the difference in the nature of the tasks requires different types of features; classification features are class-discriminitive whereas RL features are action-discriminitive, and these may be mutually exclusive. Characterizing the differences between deep classification features and deep RL features, and determining how these differences affect feature transfer, will be critical future work for enabling multi-paradigm lifelong learning. As an additional challenge, our tightly-integrated system required both the multi-task classification model and the multi-task RL policy model to be loaded in GPU memory at the same time, which significantly limited the size of the network architectures that we were able to explore using university resources. Future work will also need to place a stronger emphasis on memory-efficient lifelong learning models than in single-paradigm domains.

4.3.4 Lifelong Learning on Physical Robots.

Our above results focused on simulated evaluation of service robot tasks in AI Habitat Matterport 3D environments. While the simulated environments were constructed from data collected in real interior settings from rgbd sensors commonly used by service robots, there is still a large engineering gap that must be addressed to evaluate our integrated framework for lifelong learning on physical robot systems. Towards this goal, we have developed lifelong learning service robot infrastructure in parallel with the L2M program. We include a brief discussion of our first live deployment of a

lifelong learning service robot in a real-world environment as a case study to show one path towards realizing our framework on deployed service robots.

We conducted a live demo showcasing real-time service robot lifelong learning in front of a live audience as a demo at the ICRA 2022 conference. In order to make this feasible and robust enough to use in a human-interactable environment, we contrived a simplified evaluation protocol where instead of longer time-horizon scavenger hunt tasks, the robot needed to perform occupancy grid prediction. The goal of the task was to infer the presence of non-traversable obstacles, their geometry, and the free space around them based only on the robot's immediate observation. These features are not fully observable from the robot's on-board sensors due to limited field of view and line-of-sight occlusion. Formulating this as a lifelong learning experiment, we had people rearrange the obstacles in the environment however they wanted to challenge the robot, and treated each novel environment layout as a new prediction task. To accomplish this we developed a system that built on previous work using model ensembling, and added replay-based regularization to promote lifelong learning across a progressive series of unseen environments.

Upon facing each new arrangement of the environment, the robot would simultaneously execute the following three processes autonomously:

- Collect new training data with the new obstacle configuration
- Refine the models in the ensemble on previous batches of data from the current and previous environments
- Evaluate the current model in the previously unseen environment

For the purposes of evaluation, the ground truth data was collected via a custom ROS navigation stack, designed to not require human annotation. This enables automatic evaluation on large-scale data without human intervention. The demo ran continuously for three-hour sessions over two days during ICRA 2022, over 5-6 novel environment configurations of the environment per session, showcasing real-time lifelong learning with our service robot in front of a live audience (Figure 43). Sample results from late in the robot's lifetime are shown in Figure 44, which displays both the robot's current prediction of obstacles and free space in its immediate environment (left), and learning curves evaluating prediction accuracy for each environment across the entire lifetime (right).

4.4 Evaluation of Individual Lifelong Learning Algorithms and Approaches Used as Components within the L2M System

In this section, we describe the evaluation and results of individual lifelong learning algorithms and approaches that were used as components within the L2M system. The technical description of the corresponding algorithms are given in Section 3.3.

4.4.1 Deconvolutional factorized CNN (DF-CNN) for lifelong deep learning.

Data sets and tasks. We generated two lifelong learning problems using the CIFAR-100 [172] and Office-Home [173] data sets. For CIFAR-100, we created a series of 10 image classification

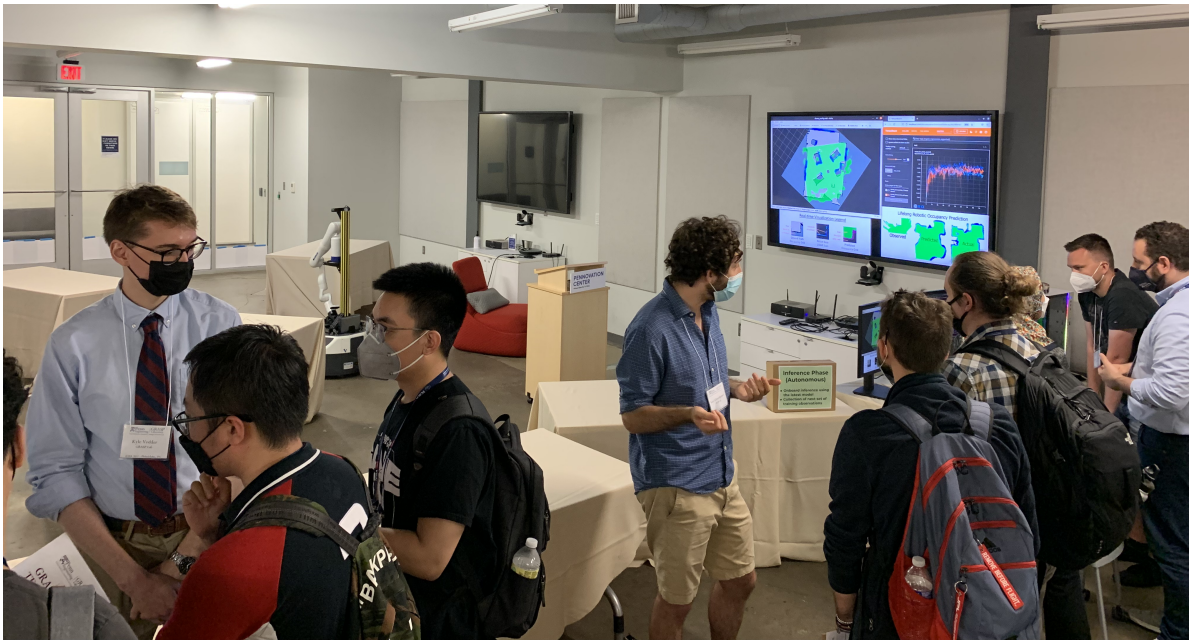


Figure 43: Demonstration of Real-Time Lifelong Learning on a Mobile Service Robot in Front of a Live Audience During ICRA 2022 in Philadelphia

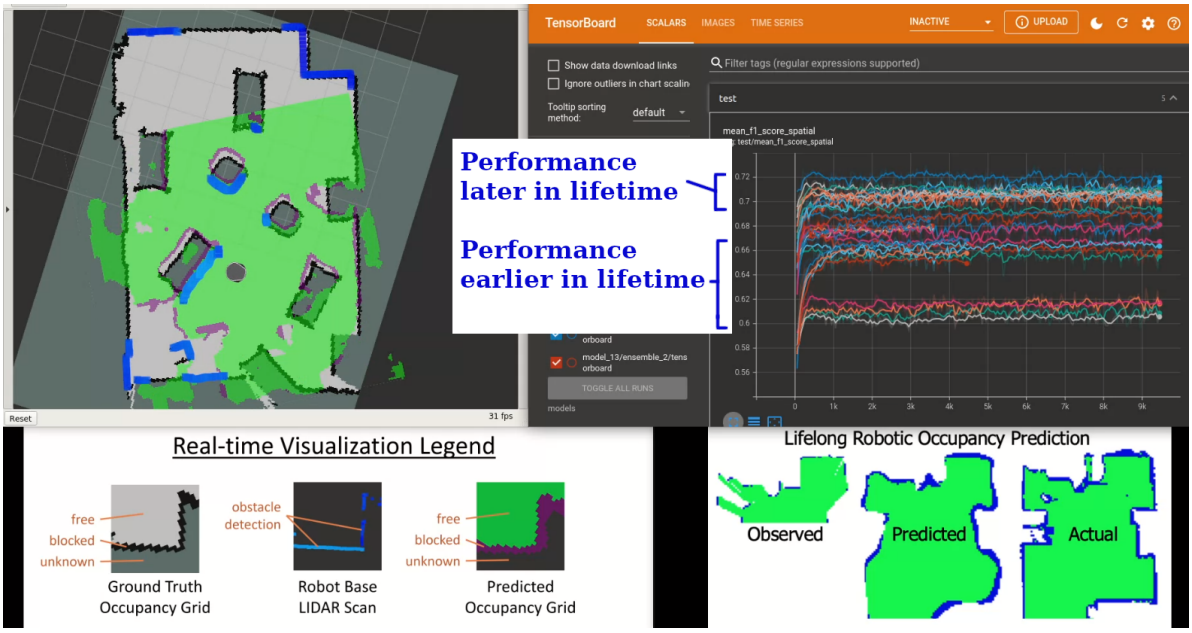
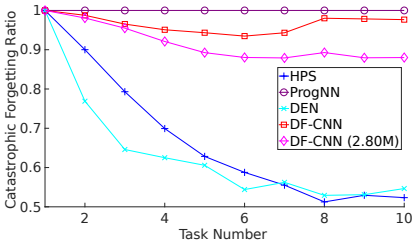


Figure 44: Real-Time Visualization of Current Occupancy Grid Prediction and Lifelong Learning Results from the Live Robot Demo

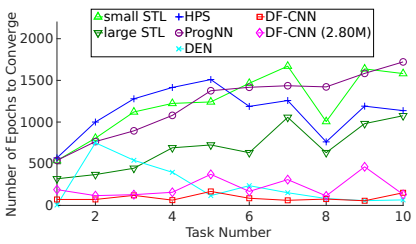
tasks, where each task consists of ten distinct classes. For each task, we sampled only 4% of the available CIFAR-100 data following a lifelong learning assumption of limited per-task training data [174], and split it into training and validation sets in the ratio 5.6:1 (170 training and 30 validation instances per task). We used all CIFAR-100 test images for the test set (1,000 instances per task).

Model	Peak Acc.	Time (10k sec)
STL (small)	31.2%±2.2	4.99±0.007
STL (large)	34.3%±1.1	5.46±0.005
HPS	24.7%±0.6	5.14±0.014
ProgNN	31.3%±1.1	9.21±0.019
DEN	30.2%±0.4	1.12±0.028
DF-CNN	37.2%±1.3	6.66±0.013
DF-CNN (2.8M)	36.2%±3.5	5.34±0.008

(a) Peak Per-Task Accuracy and Training Time with 95% Confidence Intervals



(b) Catastrophic Forgetting Ratio



(c) Speed of Convergence

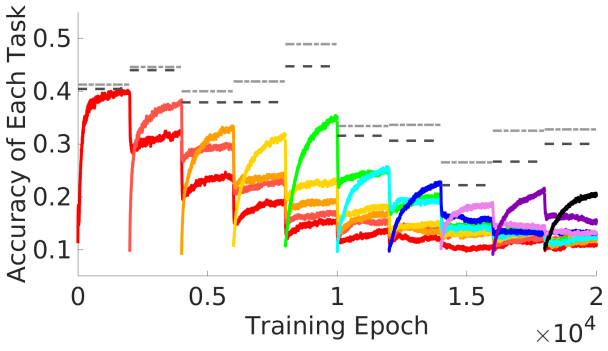
Figure 45: Performance Metrics of Models on CIFAR-100 Lifelong Learning Tasks

The Office-Home dataset is naturally split into multiple domains, and we focus on two of those domains: Product images and Real-World images. We created 5 image classification tasks from each of these two domains, resulting in 10 tasks with 13 image classes per task. There is no pre-specified training/validation/test split in the original data set, so we randomly split the data into those with a 60%, 10%, and 30% ratio, respectively. This results in approximately 550 training, 90 validation, and 250 test instances.

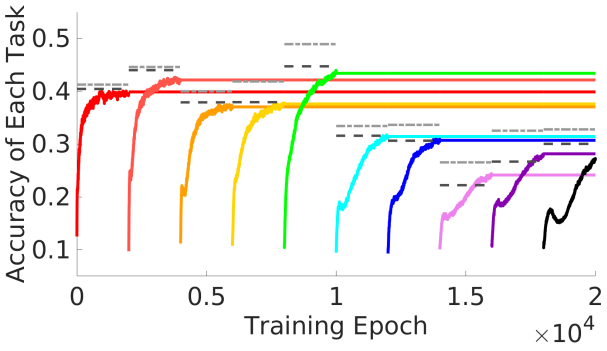
Baseline Approaches We compare DF-CNN to single-task learning and three baselines that are representative of the different approaches in lifelong learning of classification tasks:

- *Single-task learning:* STL trains a separate and isolated network for each task. STL has a clear disadvantage against transfer-based methods in the few-data or noisy-data regime.
- *Hard parameter sharing:* HPS [175] involves sharing the lower layers across tasks, with separate task-specific output layers. A useful heuristic which we follow is that all convolution layers are shared while all fully-connected layers are task-specific. HPS is expected to perform well when tasks share a common set of useful representations, but may fail when tasks are sufficiently dissimilar.
- *Progressive neural networks:* ProgNN [38] allows each task model to build upon its predecessors. Note that their paper applied the architecture to reinforcement learning, while our evaluation focuses on supervised settings. For the construction of the ProgNNs, we reduced the dimension of the previous task models' features by a factor of two (for details, see [38, Section 2: Adapters]).
- *Dynamically expandable network:* DEN [176] grows the size of the neural network according to the performance on the current task. The DEN adapts to a new task by a combination of selective retraining, expansion of the network to improve performance, and splitting to avoid catastrophic forgetting. We used the DEN implementation provided by the original authors.

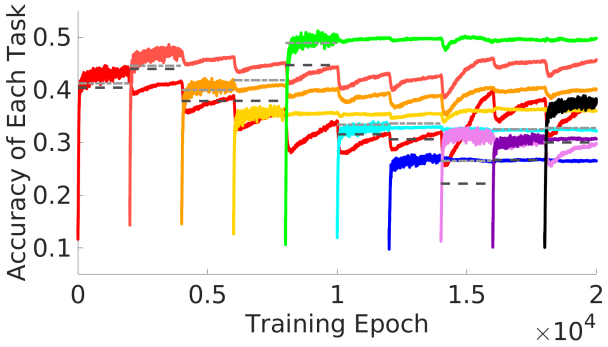
Methodology. All models were trained end-to-end on only one task at any moment, and the task was switched to the next one after every 2,000 (CIFAR-100) and 1,000 (Office-Home) training epochs, regardless of the model's convergence. The optimal hyper-parameters for each model were



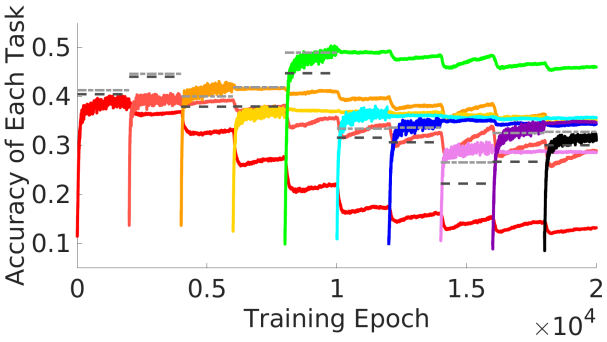
(a) Hard-Parameter Sharing (2.69M Parameters Total)



(b) Progressive Neural Net (3.51M Parameters Total)



(c) DF-CNN (7.96M Parameters Total)



(d) DF-CNN (2.80M Parameters Total)

Figure 46: Mean Test Accuracy in Lifelong Learning on CIFAR-100

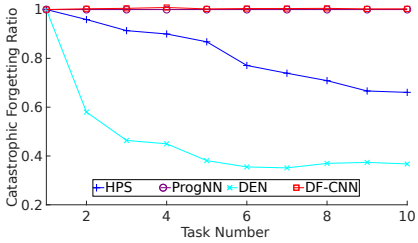
determined by performance on the validation sets. In addition to the lifelong learning baseline models, we compared two versions of STL: one with 3.28M (CIFAR-100) or 26.8M (Office-Home) parameters total (328k or 2.68M per individual task model) and the other with 9.35M (CIFAR-100) or 129.3M (Office-Home) parameters total (935k or 12.9M per task model). Our full DF-CNN has 7.96M (CIFAR-100) or 201.8M (Office-Home) parameters total, so we also examined a reduced-size DF-CNN with 2.8M parameters total in order to better match the total number of parameters of the baseline approaches in the CIFAR-100 experiments. Note that, since the purpose of the evaluation is to compare the approaches for knowledge transfer across tasks in a lifelong learning setting, the experiment design (and consequently individual model performance) may differ from other single-task-focused publications on these methods or data sets.

Metrics. We assessed performance of our DF-CNN as well as aforementioned approaches by measuring accuracy on the held-out test set for all learned tasks at each timestep. To aggregate performance across tasks, we also computed the following metrics:

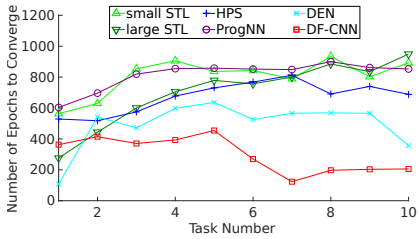
- *Peak Per-Task Accuracy:* The best test accuracy of each task during its training phase. This metric focuses on the approach's peak performance on the current task.
- *Catastrophic Forgetting Ratio:* The ratio of a task's test accuracy after training on subsequent tasks to its peak per-task accuracy. This ratio shows how much the approach can maintain its performance on older tasks.

Model	Peak Acc.	Time (10k sec)
STL (small)	45.5%±0.5	3.79±0.009
STL (large)	51.9%±0.9	6.09±0.005
HPS	52.0%±0.7	3.79±0.012
ProgNN	46.4%±1.0	11.7±0.003
DEN	31.6%±1.0	4.09±0.010
DF-CNN	49.1%±0.6	4.11±0.004

(a) Peak Per-Task Accuracy and Training Time with 95% Confidence Intervals



(b) Catastrophic Forgetting Ratio



(c) Speed of convergence

Figure 47: Performance Metrics on Office-Home Lifelong Learning Tasks

- *Convergence:* We measure the convergence of training on a task as the number of epochs over the training set needed for the test accuracy to reach 98% of its peak per-task accuracy. The number of training epochs till convergence shows the effect of knowledge transfer from previously learned tasks.

Results on Lifelong Learning The performance of all approaches is summarized in Figures 45 (CIFAR-100) and 47 (Office-Home). For the CIFAR-100 experiments, we also depict the lifelong learning process by visualizing the dynamic test accuracy of each task model over time, averaged over 5 trials, in Figure 46. Once a task has been learned, we repeatedly evaluated its model's performance as the system learns more tasks, exploring the effect of learning subsequent tasks on previous task models. Significant decreases in task performance after training indicate catastrophic forgetting [177]; increases in performance when training on other tasks indicate (positive) reverse transfer.

First, we can observe that HPS and DEN suffer from catastrophic forgetting as the shared layers were adapted to new tasks, as shown by the rapid decline in performance once learning on each task finishes (Fig. 46a as well as Fig. 45b and Fig. 47b). Additionally, both models could not achieve a peak per-task accuracy comparable to or better than that of STL consistently in the the CIFAR-100 experiments, and even HPS did not converge to peak per-task accuracy faster than STL. This means that the adaptation of the knowledge in these explicit weight-sharing models is not guaranteed to have a positive effect on training, even after the neural network shifts its attention to focus more on current tasks and forget knowledge of previous tasks.

In contrast to HPS and DEN, ProgNN is able to retain its performance on previous tasks after learning new tasks, because it is designed not to update the parameters for previous tasks. The lateral connections of ProgNN improved test accuracy in a few tasks (e.g., the 7–9th tasks of the CIFAR-100 experiment), but the benefit of transfer was marginal in comparison with the single-task learners. Moreover, ProgNN requires approximately twice as much training time as others.

DF-CNN showed significant improvement in peak per-task accuracy over STL, HPS, and ProgNN for the CIFAR-100 experiments, and peak per-task accuracy better than STL for the Office-Home experiments. Moreover, DF-CNN converges to 98% of peak per-task accuracy more than twice as fast as other approaches do. The improvement in peak per-task accuracy compared to STL, together with the improved convergence speed relative to STL, demonstrate positive knowledge transfer from older tasks within the DF-CNN framework.

The performance of previous task models within the DF-CNN deteriorated slightly as the system observed new tasks because the shared knowledge base was updated by the new tasks without consideration to previous tasks (Fig. 45b and 47b). However, the rate of losing performance is much slower than the degradation of HPS and DEN in both lifelong learning experiments, and we can even observe performance recovering over time (Fig. 46c). Especially, in the CIFAR-100 experiments (Fig. 46c), the performance of the model on the earliest tasks appears to have the most degradation, and it maintains almost constant post-training performance once the shared knowledge base became mature (e.g., the 4-10th tasks). Moreover, we can find positive reverse transfer of knowledge from new tasks to older tasks, starting at the 8th task of the CIFAR-100 experiments, which had not occurred during the training of other baseline models.

The reduced-size DF-CNN ($\sim$2.8M trainable parameters) catastrophically lost its performance on the first task because of its reduced capacity of the shared knowledge. Even with the limited capacity of both shared knowledge and task-specific knowledge transformation, the reduced-size DF-CNN still showed improvement in accuracy, speed of convergence and robust retention of performance on previous tasks as compared to the baselines. These results support the benefit of knowledge transfer between tasks through the shared knowledge and deconvolutional mapping.

In summary, the alternative approaches to lifelong learning achieved improvement in peak per-task accuracy in comparison with STL on one of two data sets, but they slowly converged to their peak performance and showed other weaknesses such as catastrophic forgetting or large training time. Contrary to this, our DF-CNN achieved good peak per-task accuracy, fast convergence, knowledge retention and reasonable training time simultaneously. This result supports that our DF-CNN is able to extract and transfer knowledge between tasks more flexibly than competing methods while resisting catastrophic forgetting in the lifelong setting.

Remark Deconvolutional factorization provides an effective means of knowledge transfer between CNNs in lifelong learning settings. Even though our DF-CNN architecture must train more parameters as compared to other approaches with the same base model (i.e., the individual task CNNs), it converges faster while providing comparable or better accuracy than competing approaches. Critically, the use of deconvolutional factorization and tensor contraction provides for flexible transfer between tasks, enabling the DF-CNN to resist catastrophic forgetting.

4.4.2 Lifelong Policy Gradients: Faster Training Without Forgetting (LPG-FTW).

Baselines We compared against STL, which does not transfer knowledge across tasks, using NPG as described in Section 3.3.2. We then chose EWC [73] from the single-model family, which places a quadratic penalty for deviating from earlier tasks' parameters. Finally, we compared against PG-ELLA [42]. All lifelong algorithms used NPG as the base learning method.

Evaluation procedure We chose the hyper-parameters of NPG to maximize the performance of STL on a single task, and used those hyper-parameters for all agents. For EWC, we searched for the regularization parameter over five tasks on each domain. For LPG-FTW and PG-ELLA, we fixed all regularization parameters to 10^{-5} and the number of columns in $\boldsymbol{L}$ to $k{=}5$, unless otherwise noted. In LPG-FTW, we used the simplest setting for the update schedule of $\boldsymbol{L}$, $M{=}N$. All experiments were repeated over five trials with different random seeds for parameter initialization and task ordering.

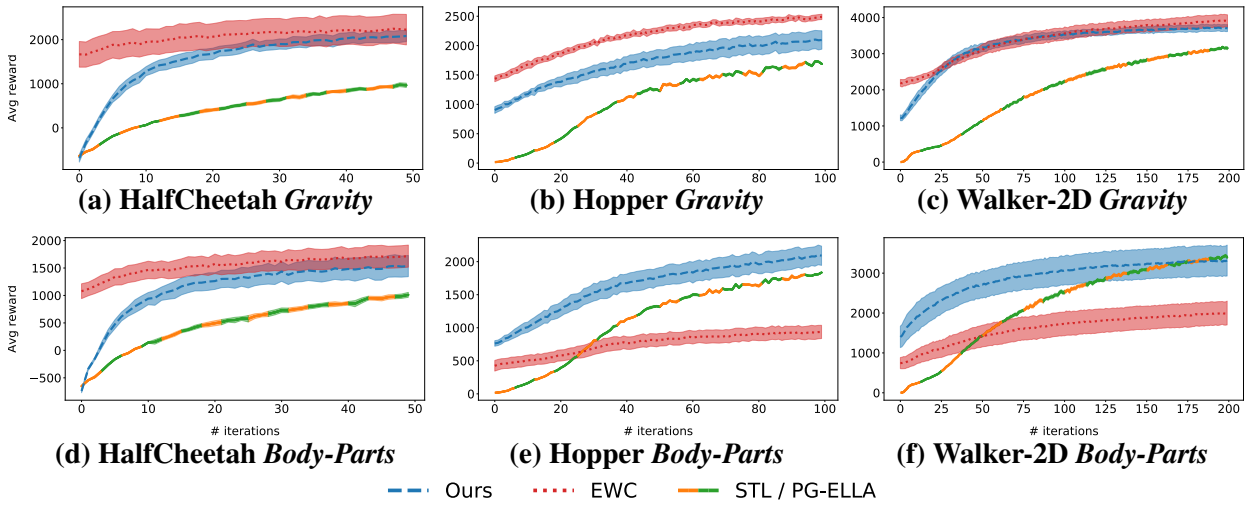


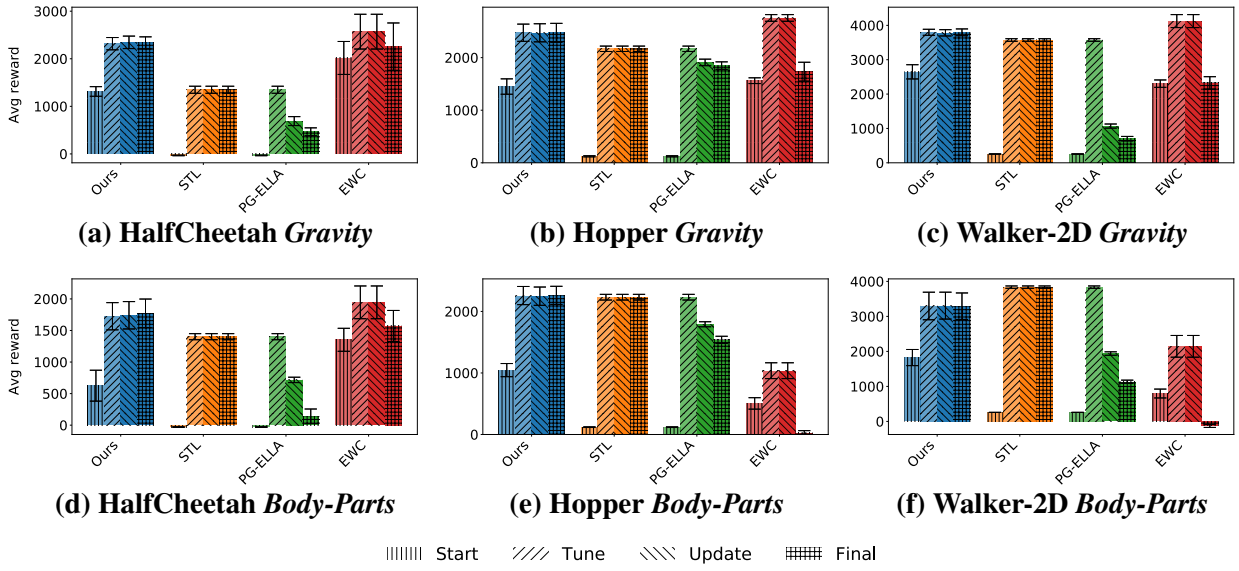
Figure 48: Average Performance During Training Across All Tasks for Six MuJoCo Domains

Empirical evaluation on OpenAI Gym MuJoCo domains We first evaluated LPG-FTW on simple MuJoCo environments from OpenAI Gym [111, 178]. We selected the HalfCheetah, Hopper, and Walker-2D environments, and created two different evaluation domains for each: a *gravity* domain, where each task corresponded to a random gravity value between $0.5g$ and $1.5g$, and a *body-parts* domain, where the size and mass of each of four parts of the body (head, torso, thigh, and leg) was randomly set to a value between $0.5\times$ and $1.5\times$ its nominal value. These choices led to highly diverse tasks. We generated tasks using the `gym-extensions` [179] package, but modified it so each body part was scaled independently.

We created $T_{\text{max}} = 20$ tasks for HalfCheetah and Hopper domains, and $T_{\text{max}} = 50$ for Walker-2D domains. The agents were allowed to train on each task for a fixed number of iterations before moving on to the next. For these simple experiments, all agents used linear policies. For the Walker-2D *body-parts* domain, we set the capacity of $\boldsymbol{L}$ to $k = 10$, since we found empirically that it required a higher capacity. The NPG hyper-parameters were tuned without body-parts or gravity modifications.

Figure 48 shows the average performance over all tasks as a function of the NPG training iterations. LPG-FTW consistently learned faster than STL, and obtained higher final performance on five out of the six domains. Learning the task-specific coefficients $\boldsymbol{s}^{(t)}$ directly via policy search increased the learning speed of LPG-FTW, whereas PG-ELLA was limited to the learning speed of STL, as indicated by the shared learning curves. EWC was faster than LPG-FTW in reaching high-performing policies in four domains, primarily due to the fact that EWC uses a single shared policy across all tasks, which enables it to have starting policies with high performance. However, EWC failed to even match the STL performance in two of the domains. We hypothesize that this was due to the fact that the tasks were highly varied (particularly in the *body-parts* domains, since there were four different axes of variation), and the single shared policy was unable to perform well in all domains.

Results in Figure 48 consider only how fast the agent learns a new task using information from earlier tasks. PG-ELLA and LPG-FTW then perform an update step (Eq. 4 for LPG-FTW) where they incorporate knowledge from the current task into $\boldsymbol{L}$. The third bar from the left per each



(a) HalfCheetah *Gravity* **(b) Hopper *Gravity*** **(c) Walker-2D *Gravity***

(d) HalfCheetah *Body-Parts* **(e) Hopper *Body-Parts*** **(f) Walker-2D *Body-Parts***

Figure 49: Average Performance at the Beginning of Training (Start), After All Training Iterations (Tune), After the Update Step for PG-ELLA and LPG-FTW (Update), and After All Tasks Have Been Trained (Final)

algorithm in Figure 49 shows the average performance after this step, revealing that LPG-FTW maintained performance, whereas PG-ELLA's performance decreased. This is because LPG-FTW ensures that the approximate objective is computed near points in the parameter space that the current basis $\boldsymbol{L}$ can generate, by finding $\boldsymbol{\alpha}^{(t)}$ via a search over the span of $\boldsymbol{L}$. A critical component of lifelong learning algorithms is their ability to avoid catastrophic forgetting. To evaluate the capacity of LPG-FTW to retain knowledge from earlier tasks, we evaluated the policies obtained from the knowledge base $\boldsymbol{L}$ trained on all tasks, without modifying the $\boldsymbol{s}^{(t)}$'s. The rightmost bar in each algorithm in Figure 49 shows the average final performance across all tasks. LPG-FTW successfully retained knowledge of all tasks, showing no signs of catastrophic forgetting on any of the domains. The PG-ELLA baseline suffered from forgetting in all domains, and EWC in all but one of the domains. Moreover, the final performance of LPG-FTW was the best among all baselines in all but one domain.

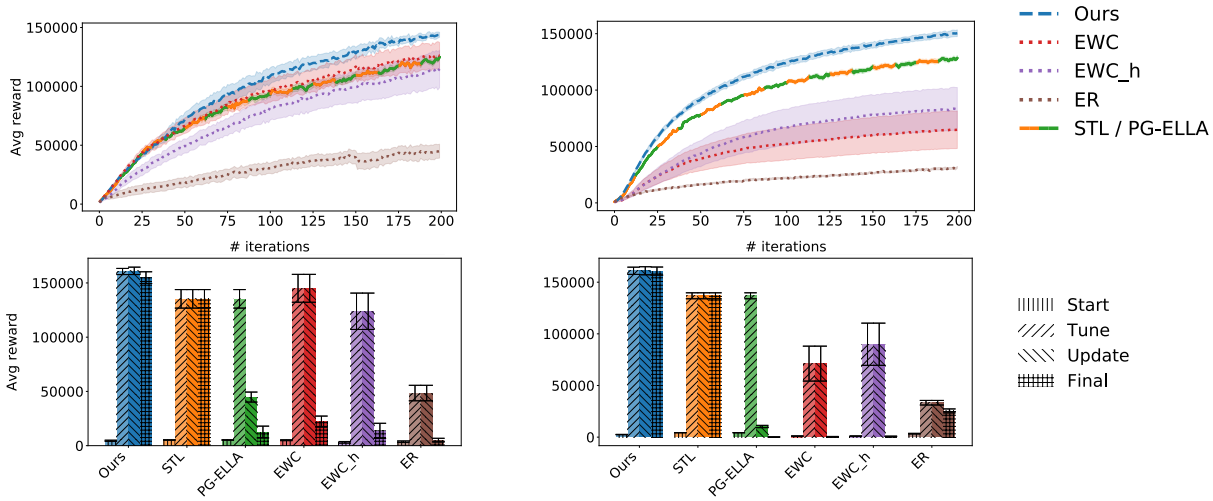
Empirical evaluation on more challenging Meta-World domains Results so far show that our method improves performance and completely avoids forgetting in simple settings. To showcase the flexibility of our framework, we evaluated it on Meta-World [180], a substantially more challenging benchmark, whose tasks involve using a simulated Sawyer robotic arm to manipulate various objects in diverse ways, and have been shown to be notoriously difficult for state-of-the-art multi-task learning and meta-learning algorithms. Concretely, we evaluated all methods on sequential versions of the MT10 benchmark, with $T_{\mathsf{max}} = 10$ tasks, and the MT50 benchmark, using a subset of $T_{\mathsf{max}} = 48$ tasks. We added an experience replay (ER) baseline that uses importance sampling over a replay buffer from all previous tasks' data to encourage knowledge retention, with a 50-50 replay rate as suggested by [154]. We chose the NPG hyper-parameters on the `reach` task, which is the simplest task from the benchmark. For LPG-FTW and PG-ELLA, we fixed the number of latent components as $k = 3$. All algorithms used a Gaussian policy parameterized by a multi-layer perceptron with two hidden layers of 32 units and $\tanh$ activation. We also evaluated EWC with

a higher capacity (EWC_h), with 50 hidden units for MT10 and 40 for MT50, to ensure that it was given access to approximately the same number of parameters as our method. Given the high diversity of the tasks considered in this evaluation, we allowed all algorithms to use task-specific output layers, in order to specialize policies to each individual task.

The top row of Figure 50 shows average learning curves across tasks. LPG-FTW again was faster in training, showing that the restriction that the agent only train the $\boldsymbol{s}^{(t)}$'s for each new task does not harm its ability to solve complex, highly diverse problems. The difference in learning speed was particularly noticeable in MT50, where single-model methods became saturated. To our knowledge, this is the first time lifelong transfer has been shown on the challenging Meta-World benchmark. The bottom row of Figure 50 shows that LPG-FTW suffered from a small amount of forgetting on MT10. However, on MT50, where $\boldsymbol{L}$ trained on sufficient tasks for convergence, our method suffered from no forgetting. In contrast, none of the baselines was capable of accelerating the learning, and they all suffered from dramatic forgetting, particularly on MT50, when needing to learn more tasks. Adding capacity to EWC did not substantially alter these results, showing that our method's ability to handle highly varied tasks does not stem from its higher capacity from using k factors, but instead to the use of *different* models for each task by intelligently combining the shared latent policy factors.

4.4.3 Meta-Optimizer for Fast Adaptation (KFO).

We verify the the efficacy of our meta-optimizers on real-world datasets. We first show that adding linear layers ("LinNet") as a general strategy of increasing the depth of the models improves meta-learning in both shallow and deep models. To further clarify the roles of the upper layers in deep models for meta-learning, we propose a new algorithm for meta-learning, META-KFO, in §3.3.3. While META-KFO is primarily used in this work for investigating how MAML works, it also attains state-of-the-art performances.



(a) Meta-World MT10 **(b) Meta-World MT50**

Figure 50: Performance on the Meta-World Benchmark: (Top) During Training, and (Bottom) at the Beginning of Training (Start), After All Training Iterations (Tune), After the Update Step for PG-ELLA and LPG-FTW (Update), and After All Tasks Have Been Trained (Final)

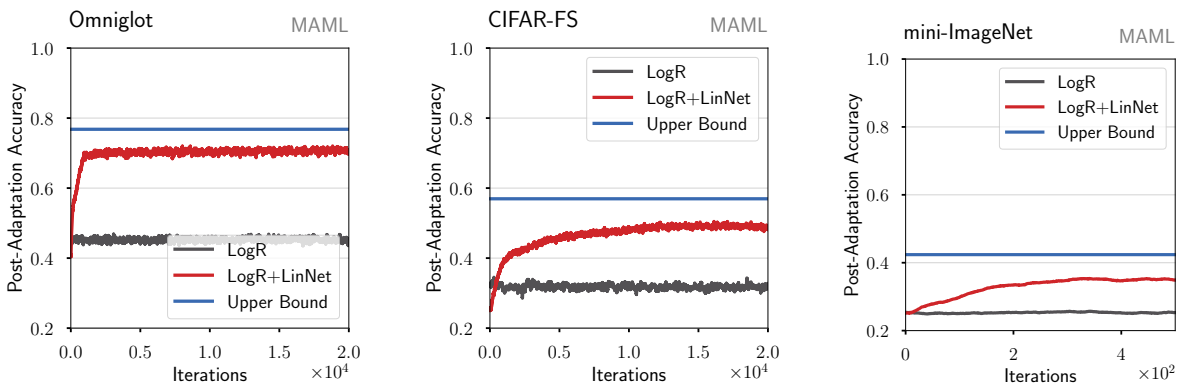


Figure 51: Meta-Training Logistic Regression Models with MAML on Omniglot, CIFAR-FS, and mini-ImageNet.

Table 8: Accuracy Improves by Adding Linear Layers

Method	MAML				MAML w/ LinNet			
CNN Layers	2	3	4	6	2	3	4	6
Omniglot	66.8	93.5	98.5	97.6	88.1	95.5	98.1	97.6
CIFAR-FS	62.2	68.9	70.9	71.3	66.1	71.1	74.4	71.9
mini-ImageNet	52.6	54.0	64.1	64.6	60.5	60.2	64.9	64.1

Datasets and settings Throughout our study, we use the standard 5-ways and 5-shots setting on the Omniglot [181], CIFAR-FS [182], and mini-ImageNet [183] datasets. We denote by CNN(X) the convolutional network with X convolutional layer; for example, CNN(4) corresponds to the baseline network also used in [184] and [185] among many others. To ensure fair comparison, we independently reimplemented each algorithmic variant and found the best hyper-parameter values for each architecture-algorithm pair via grid-search.

Linear layers improve shallow linear models Figure 51 displays the results of meta-learning using MAML on two standard benchmark datasets with logistic/softmax regression models. The light blue horizontal lines denote the best performance if the models are trained with sufficient data on the meta-testing tasks. The black lines are the meta-learning performance, which only slightly improve upon chance levels. (20% on Ominglot and CIFAR-FS) However, when linear layers are added to these linear models, the meta-learning performance is significantly improved. (72% on Omniglot, 48% on CIFAR-FS)

Linear layers improve deep nonlinear models Table 8 lists the positive results of adding two linear layers to different CNN architectures. When the number of CNN layers is less than 6, the addition improves meta-learning performances. At CNN(6), there are degradations in performance by the original MAML on Omniglot and CIFAR-FS such that LinNet does not improve further. On mini-ImageNet, while MAML improves, LinNet decreases though its performance at CNN(4) is still the best.

Table 9: Meta-Optimizers Outperform MAML on CNN(2)

Dataset	MAML	MAML w/			
		MSGD	MC	T-Nets	META-KFO
Omniglot	66.6	74.07	94.63	92.27	**96.62**
CIFAR-FS	62.2	62.82	68.37	66.42	**69.64**
mini-ImageNet	52.6	**59.90**	58.95	58.47	59.08

Comparing Meta-Optimizers Table 9 contrasts different approaches for improving meta-learning by MAML on CNN(2) *without* increasing the size of the model after adaptation. All methods improve the original MAML while META-KFO improves the most on Omniglot and CIFAR-FS. On mini-ImageNet, all methods improve about the same amount. We also compared the meta-optimizers when ANIL is used to meta-learn. Again, all methods improve the baseline ANIL and META-KFO improves most significantly.

4.4.4 Occupancy Anticipation for Efficient Exploration and Navigation (OCCANT).

In the following experiments we demonstrate that 1) our occupancy anticipation module can successfully infer unseen parts of the map and 2) trained together with an exploration and navigation policy, it accelerates active mapping and navigation in new environments.

Experimental setup We use the Habitat [31] simulator along with Gibson [186] and Matterport3D [187] environments. Each dataset contains around 90 challenging large-scale photo-realistic 3D indoor environments such as houses and office buildings. On average, the Matterport3D environments are larger. Our observation space consists of 128×128 RGB-D observations and odometry sensor readings that denote the change in the agent's pose x, y, θ. Our action space consists of three actions: MOVE-FORWARD by 25cm, TURN-LEFT by $10°$, TURN-RIGHT by $10°$. For navigation, we add a STOP action, which the agent emits when it believes it has reached the goal. We simulate noisy actuation and odometer readings for realistic evaluation.

We train our exploration models on Gibson, and then transfer them to PointGoal navigation on Gibson and exploration on Matterport3D. We use the default train/val/test splits provided for both datasets [31] with disjoint environments across the splits. For evaluation on Gibson, we divide the validation environments into small (area less than 36m^2) and large (area greater than 36m^2) to observe the influence of environment size on results. For policy learning, we use the Adam optimizer and train on episodes of length 1000 for $1.5 - 2$ million frames of experience. Please see [29] for more details.

Baselines: We define baselines based on prior work:

- **ANS(rgb)** [47]: This is the state-of-the-art Active Neural SLAM approach for exploration and navigation. We use the original mapper architecture [47], which infers the visible occupancy from RGB.[8]

[8] We use our own implementation of ANS since authors' code was unavailable at the time of our experiments.

Table 10: Occupancy Anticipation Results on the Gibson Validation Set

Method	IoU %			F1 score %		
	free	occ.	mean	free	occ.	mean
all-free	30.1	0	15.1	43.6	0	21.8
all-occupied	0	25.1	12.6	0	39.2	19.6
ANS(rgb)	12.1	14.9	13.5	19.6	24.9	22.5
ANS(depth)	14.5	24.1	19.3	23.1	37.6	30.4
View-extrap.	15.5	26.4	21.0	25.0	40.4	32.7
OccAnt(rgb)	44.4	47.9	46.1	58.2	62.9	60.6
OccAnt(depth)	50.4	**61.9**	56.1	63.8	**75.0**	69.4
OccAnt(rgbd)	**51.5**	61.5	**56.5**	**64.9**	74.8	**69.8**

- **ANS(depth)**: We use depth projection to infer the visible occupancy (similar to [48]) instead of predicting it from RGB.
- **View-extrap.**: We extrapolate an 180° FoV depth map from 90° FoV RGB-D and project it to the top-down view. This is representative of scene completion approaches [16, 188].
- **OccAnt(GT)**: This is an upper bound that cheats by using the ground-truth anticipation maps for exploration and navigation.

We implement all baselines on top of the ANS framework. Our goal is to show the impact of our occupancy model, while fixing the backbone navigation architecture and policy learning approach across methods for a fair comparison. We consider three versions of our models based on the input modality:

- **OccAnt(depth)**: anticipate occupancy given the visible occupancy map.
- **OccAnt(rgb)**: anticipate occupancy given only the RGB image. We replace the depth projections in Figure 6 with the pre-trained ANS(rgb) estimates (kept frozen throughout training).
- **OccAnt(rgbd)**: anticipate occupancy given the full RGB-D inputs.

By default, our methods use the proposed anticipation reward in Eqn. 17. We denote ablations without this reward as "w/o AR".

Occupancy anticipation results First we evaluate the per-frame prediction accuracy of the mapping models trained during exploration. We evaluate on a separate dataset of images sampled from validation environments in Gibson at uniform viewpoints from discrete locations on a 1m grid, a total of $1,034$ (input, output) samples. This allows standardized evaluation of the mapper, independent of the exploration policy.

To quantify the local occupancy maps' accuracy, we compare the predicted maps to the ground truth. We report the Intersection over Union (IoU) and F1 scores for the "free" and "occupied" classes

independently. In addition to the baselines from the previous section, we add two naive baselines that classify all locations as free (all-free), or occupied (all-occupied).

Table 10 shows the results. Our models, OccAnt(·), substantially improve the map quality and extent, showing the advantage of learning to anticipate 3D structures beyond those directly observed. Our anticipation models OccAnt are substantially better than all the baselines. Comparing different modalities, OccAnt(depth) is much better than OccAnt(rgb) under all the metrics. This makes sense, as visible occupancy is directly computable from the depth input, but must be inferred for RGB (see Fig. 52). Interestingly, the rgbd models are not better than the depth-only models, likely because (1) geometric cues are more easily learned from depth than RGB, and (2) the RGB encoder contains significantly more parameters and could lead to overfitting. See Table S5 in Supp. for network sizes. Overall, Table 10 demonstrates our occupancy anticipation models successfully broaden the coverage of the map beyond the visible regions.

Exploration results Next we deploy our models for visual exploration. The agent is given a limited time budget (T=1000) to intelligently explore and build a 2D top-down occupancy map of a previously unseen environment.

To quantify exploration, we measure both map quality and speed (number of agent actions): (1) **Map accuracy (m^2):** the area in the global map built during exploration (both free and occupied) that matches with the ground-truth layout of the environment. The map is built using predicted occupancy maps which are registered using estimated pose (may be noisy). Note that this is an unnormalized accuracy measure (see Eqn. 16). (2) **IoU:** the intersection over union between that same global map and the ground-truth layout of the environment. (3) **Area seen (m^2):** the amount of free and occupied regions *directly seen* during exploration. The map for this metric is built using ground-truth pose and depth-projections (similar to [48, 47]). (4) **Episode steps:** the number of actions taken by the agent. While the first two metrics measure the quality of the created map, the latter two are a function of how (and how long) the agent moved to get that map. Higher accuracy in fewer steps or lower area-seen is better.

All agents are trained on 72 scenes from Gibson under noisy odometry and actuation, and evaluated on Gibson and Matterport3D under both noisy and noise-free conditions.

Figure 52 shows the per-frame local occupancy predictions. In this figure, ANS(*) are restricted to only predicting occupancy for visible regions. View-extrap. extrapolates, but is unable to predict occupancy for occluded regions (first row of the figure) and struggles to make correct predictions in cluttered scenes (second row). Our model successfully anticipates with either RGB or depth. For example, in the first row, we successfully predict the presence of a corridor and another room on the left. In the second row, we successfully predict the presence of navigable space behind the table. In the third row, we are able to correctly anticipate the free space behind the chair and the corridor to the right.

Figure 54 shows the exploration results. Our approach generally outperforms the baselines, improving the map quality more rapidly, whether in terms of time (top row) or area seen (bottom row). On the top of Figure 54, our OccAnt approach rapidly attains higher map accuracy than the baselines (dotted lines). On the bottom, OccAnt achieves higher map accuracy for the same area seen (we show the best variants here to avoid clutter). These results show the agent *actively moves better* to explore the environment with occupancy anticipation. When compared on a same-modality basis, we see that OccAnt(rgb) converges much faster than ANS(rgb). Similarly, OccAnt(depth) is able to

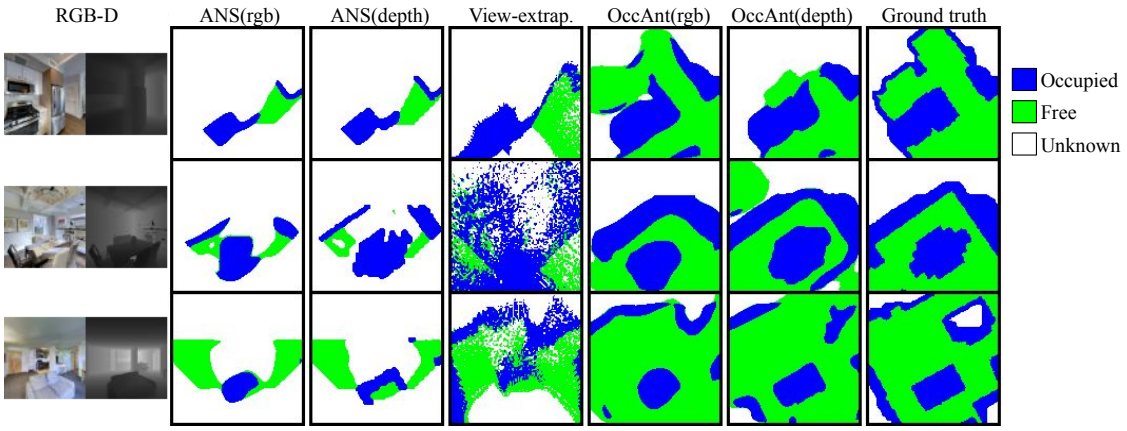


Figure 52: Per-frame local occupancy predictions. First and last columns show the RGB-D input and anticipation ground-truth, respectively.

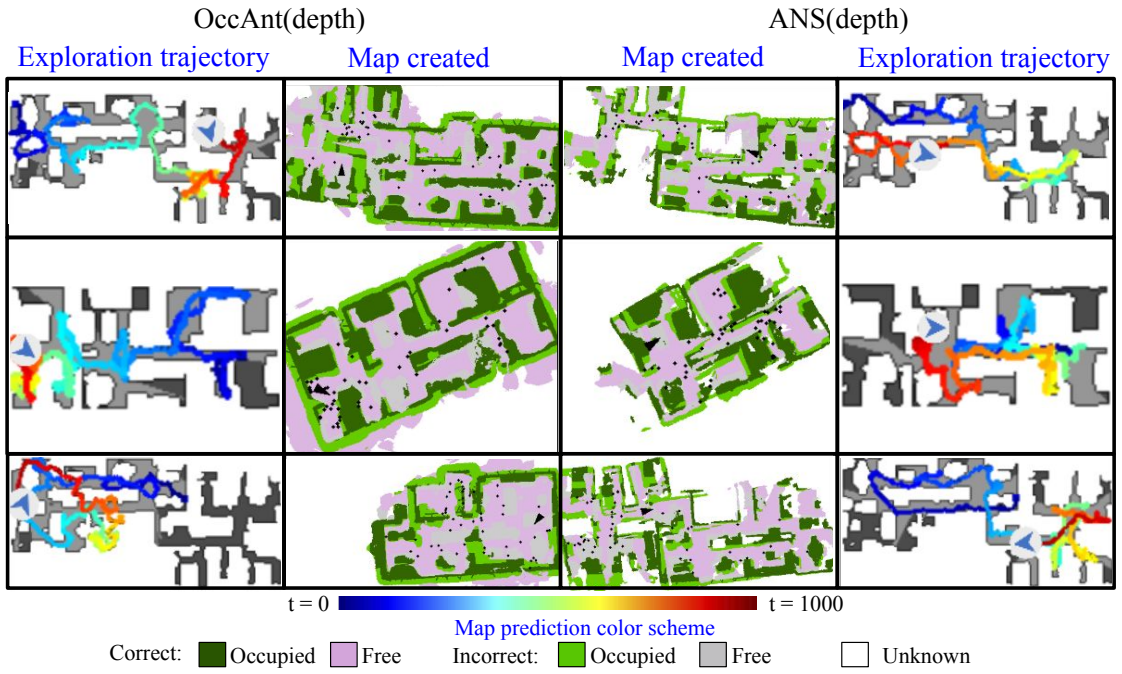


Figure 53: Exploration examples: OccAnt compared with ANS in Gibson under noisy actuation and odometry. The exploration trajectories and the corresponding maps are shown at the extremes and center, respectively.

rapidly improve the map quality and outperforms ANS(depth) on all cases. This apples-to-apples comparison shows that anticipating occupancy leads to much more efficient mapping in unseen environments. Again, using depth generally provides more reliable mapping than pure RGB.

Furthermore, the proposed anticipation reward generally provides significant benefits to map accuracy in the noisy setting (compare our full model to the "w/o AR" models in Figure 54). While map accuracy generally increases over time for noise-free conditions (see Sec. S1 in Supp.), it sometimes saturates early or even declines slightly over time in the noisy setting as noisy pose estimates accumulate and hurt map registration accuracy. This is most visible in Gibson small (top

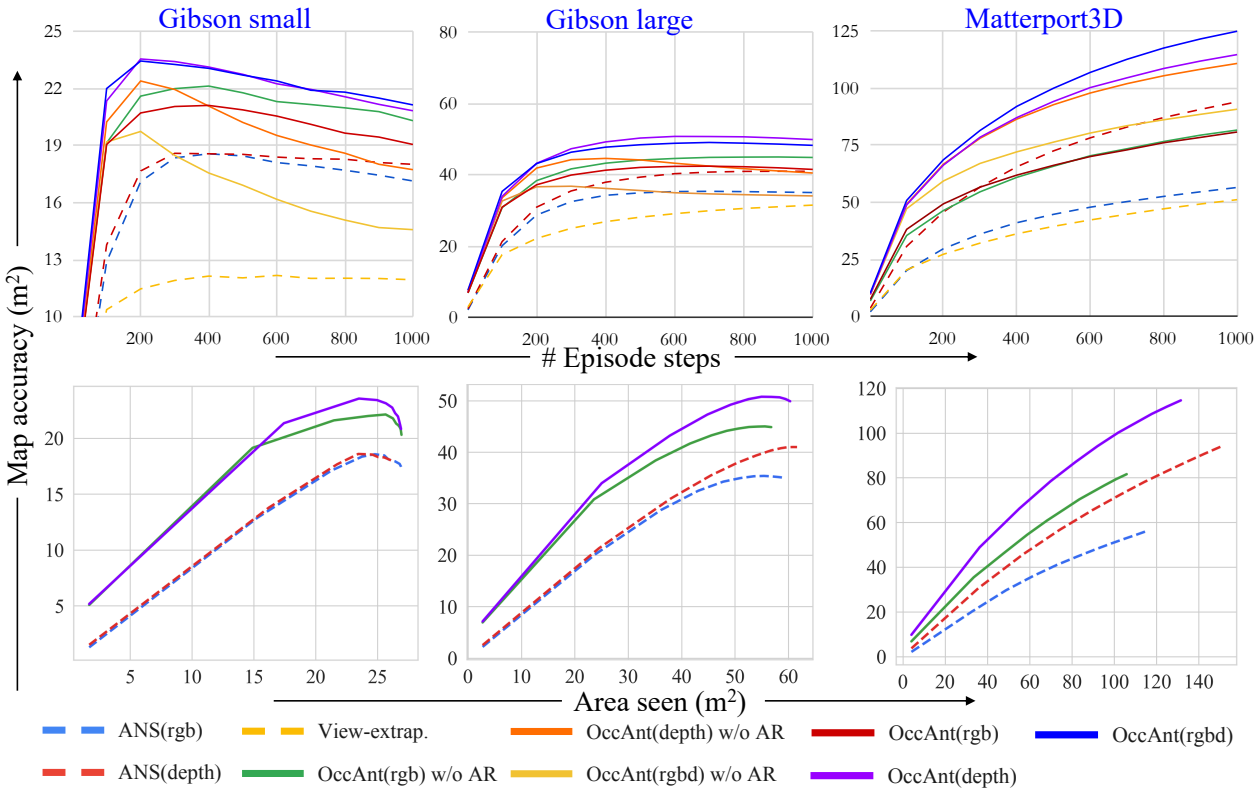


Figure 54: Exploration results: Map accuracy (m^2) as a function of episode duration (top row) and area seen (bottom row) for Gibson (small and large splits) and Matterport3D under noisy conditions (see Sec. S1 in Supp. for noise-free). Higher and steeper curves are better.

Table 11: Timed Exploration Results

	Noisy test conditions						Noise-free test conditions					
	Gibson small		Gibson large		Matterport3D		Gibson small		Gibson large		Matterport3D	
Method	Map acc.	IoU	Map acc.	IoU	Map acc.	IoU	Map acc.	IoU	Map acc.	IoU	Map acc.	IoU
ANS(rgb) [47]	18.5	55	35.0	47	44.7	18	22.4	76	43.4	64	53.4	23
ANS(depth)	18.5	56	39.4	53	72.5	26	21.4	74	48.0	72	85.9	34
View-extrap.	12.0	26	28.1	27	39.4	14	12.1	27	26.5	27	33.9	13
OccAnt(rgb) w/o AR	21.8	66	44.2	57	65.8	23	22.6	71	45.2	60	64.4	24
OccAnt(depth) w/o AR	20.2	58	44.2	54	92.7	29	**24.9**	**84**	**54.1**	**75**	**. 104.7**	**38.**
OccAnt(rgbd) w/o AR	16.9	45	35.6	40	76.3	23	24.8	**84**	52.0	71	98.7	34
OccAnt(rgb)	20.9	62	42.1	54	66.2	22	22.3	70	43.5	58	64.4	22
OccAnt(depth)	**22.7**	**71**	**50.3**	**67**	94.1	**33**	24.8	83	53.1	74	96.5	35
OccAnt(rgbd)	**22.7**	**71**	48.4	62	**99.9**	32	24.5	82	51.0	69	100.3	34
OccAnt(GT)	21.7	67	51.9	63	-	-	26.1	93	65.4	91	-	-

left plot). However, our anticipatory reward alleviates this decline.

Table 11 summarizes the map accuracy and IoU for all methods at T=500. Our method obtains significant improvements, supporting our claim that occupancy anticipation accelerates exploration and mapping. Additionally, perfect anticipation with the OccAnt(GT) model gives comparably

Table 12: PointNav Results

Method	SPL %	Success %	Time taken
ANS(rgb) [47]	66.8	87.9	254.109
ANS(depth)	76.8	86.6	226.161
View-extrap.	10.4	33.3	835.556
OccAnt(rgb)	71.2	88.2	223.411
OccAnt(depth)	77.8	91.3	194.751
OccAnt(rgbd)	**80.0**	**93.0**	**171.874**
OccAnt(GT)	89.5	96.0	125.018

Table 13: Habitat Challenge 2020 Results

	Test standard			Test challenge		
Rank	Team	SPL %	Success %	Team	SPL %	Success %
1	**OccupancyAnticipation**	19.2	24.8	**OccupancyAnticipation**	20.9	27.5
2	ego-localization [189]	10.4	13.6	ego-localization [189]	14.6	19.2
3	Information Bottleneck	5.0	7.5	DAN [190]	13.2	25.3
4	cogmodel_team	0.8	1.3	Information Bottleneck	6.0	8.8
5	UCULab	0.5	0.8	cogmodel_team	0.7	1.2
6	Habitat Team (DD-PPO) [191]	0.0	0.2	UCULab	0.1	0.2

good noisy exploration, and good gains in noise-free exploration (+10-20$\%$ IoU). This shows that there is indeed a lot of mileage in anticipating occupancy; our model moves the state-of-the-art towards this ceiling. Figure 53 shows example exploration trajectories and the final global map predictions on Gibson. In Figure 53 row 1, both methods cover similar area, but our method better anticipates the unseen parts with fewer registration errors. In row 2, our method achieves better area coverage and mapping quality whereas the baseline gets stuck in a small room for extended periods of time. Row 3 shows a failure case for our method, where it gets stuck in one part of the house after anticipating that a narrow corridor leading to a different room was occupied.

Navigation results Next we evaluate the utility of occupancy anticipation for quickly reaching a target. In PointNav [52, 53], the agent is given a 2D coordinate (relative to its position) and needs to reach that target as quickly as possible. Following [47], we use noise-free evaluation and directly transfer the mapper, planner, and local policy learned during exploration to this task. In this way, instead of navigating to a point specified by the global policy, the agent has to navigate to a fixed goal location. To evaluate navigation, we use the standard metrics—success rate, success rate normalized by inverse path length (SPL) [53], and time taken. The agent succeeds if it stops within 0.2m of the target under a time budget of $T = 1000$.

Table 12 shows the navigation results on the Gibson validation set. Our approach outperforms the baselines. Thus, not only does occupancy anticipation successfully map the environment, but it also allows the agent to move to a specified goal more quickly by modeling the navigable spaces. This apples-to-apples comparison shows that our idea improves the state of the art for PointNav. As with exploration, using ground truth (GT) anticipation leads to good gains in the navigation performance, and our methods bridge the gap between the prior state of the art and perfect anticipation.

In concurrent work, the DD-PPO approach [191] obtains 0.96 SPL for PointNav, but it requires

2.5 billion frames of experience to do so (and it fails for noisy conditions; see below). To achieve the performance of our method (0.8 SPL in 2M frames), DD-PPO requires more than 50× the experience. Our sample efficiency can be attributed to explicit mapping along with occupancy anticipation.

Finally, we validate our approach on the 2020 Habitat PointNav Challenge [192], which requires the agent to adapt to noisy RGB-D sensors and noisy actuators, and to operate without an odometer. This presents a much more difficult evaluation setup than past work which assumes perfect odometry as well as noise-free sensing and actuation [31, 47, 191]. See [29] for more details. Table 13 shows the results. Our method won the challenge, outperforming the competing approaches by large margins. While our approach generalizes well to this setting, DD-PPO [191] fails (0 SPL) due to its reliance on perfect odometry.

4.4.5 Learning Intrinsic Rewards for Policy Gradient Methods.

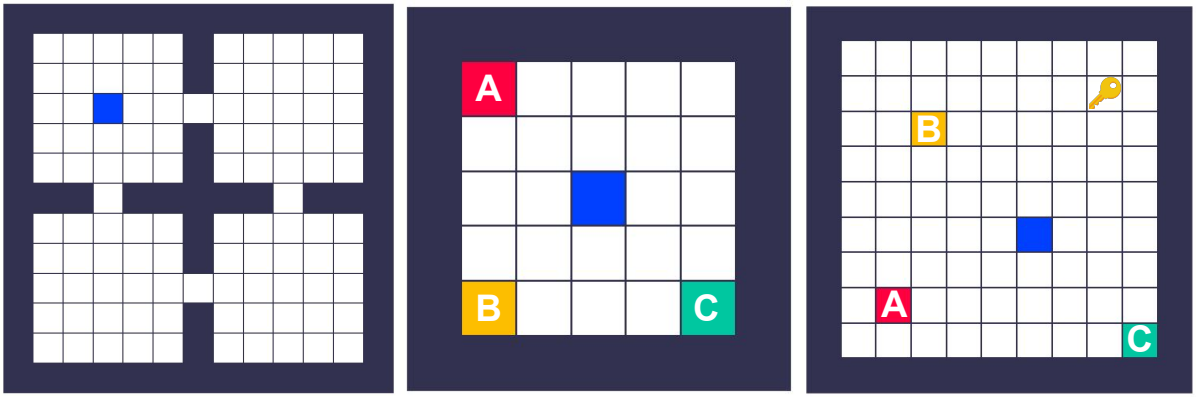
We present the results from our empirical investigations in two sections. In this section, the experiments and domains are designed to answer the following research questions:

- What kind of knowledge is learned by the intrinsic reward?
- How does the distribution of tasks influence the intrinsic reward?
- What is the benefit of the lifetime return objective over the episode return?
- When is it important to provide the lifetime history as input to the intrinsic reward?

Experimental Setup We investigate these research questions in the grid-world domains illustrated in Figure 55. For each domain, we trained an intrinsic reward function across many lifetimes and evaluated it by training an agent using the learned reward. We implemented the following baselines.

- Extrinsic-EP: A policy is trained with extrinsic rewards to maximise the episode return.
- Extrinsic-LIFE: A policy is trained with extrinsic rewards to maximise the lifetime return.
- Count-based [193]: A policy is trained with extrinsic rewards and count-based exploration bonus rewards.
- ICM [194]: A policy is trained with extrinsic rewards and curiosity rewards based on an inverse dynamics model.

Note that these baselines, unlike the learned intrinsic rewards, do not transfer any knowledge across different lifetimes. In the following subsections, we focus on analysing what kind of knowledge is learned by the intrinsic reward depending on the nature of environments, and we discuss the benefit of using the lifetime return and considering the lifetime history when learning the intrinsic reward in. The details of implementation and hyperparameters are described in the supplementary material of the original publication.



(a) Empty Rooms **(b) ABC** **(c) Key-Box**

Figure 55: Illustration of Domains for RL with Intrinsic Rewards. (a) The agent needs to find the goal location which gives a positive reward, but the goal is not visible to the agent. (b) Each object (A, B, and C) gives rewards. (c) The agent is required to first collect the key and visit one of the boxes (A, B, and C) to receive the corresponding reward. All objects are placed in random locations before each episode.

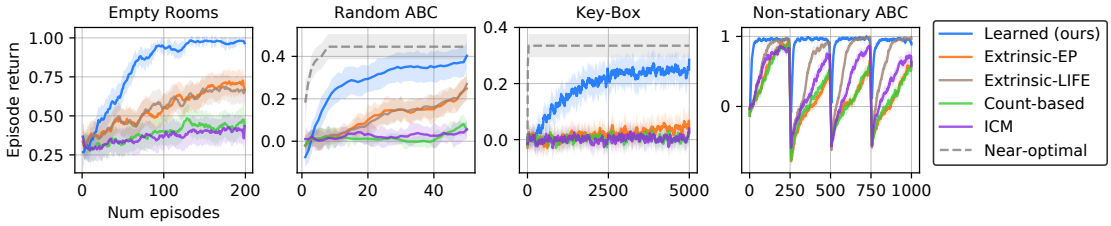
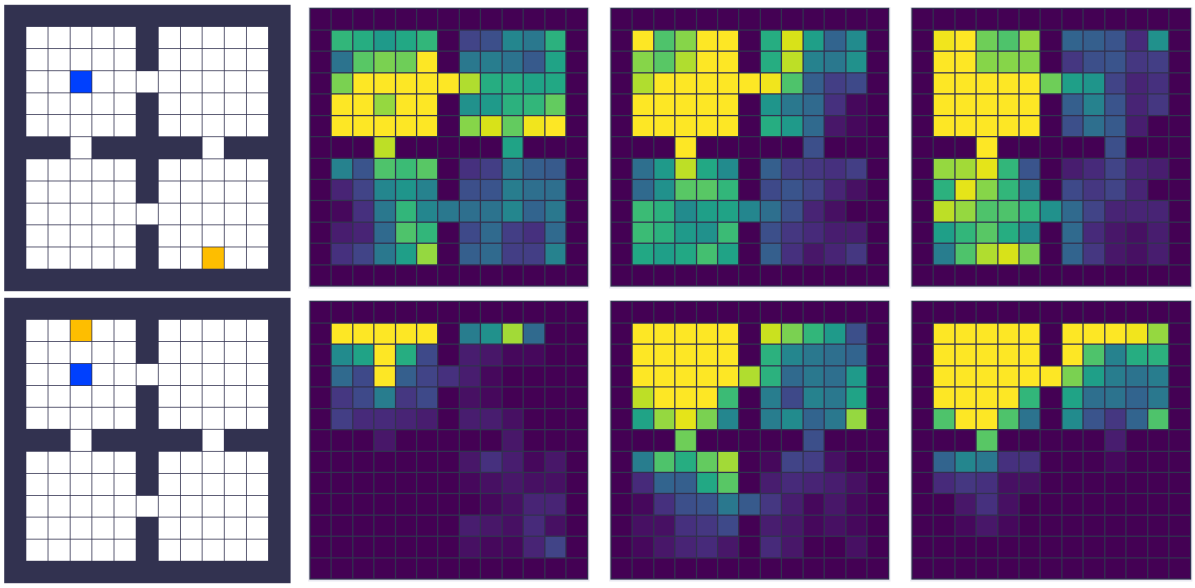


Figure 56: Evaluation of Different Reward Functions Averaged over 30 Seeds. The learning curves show agents trained with our intrinsic reward (blue), with the extrinsic reward using the episodic return objective (orange) or the lifetime return objective (brown), and with a count-based exploration reward (green). The dashed line corresponds to a hand-designed near-optimal exploration strategy.

Exploring Uncertain States We designed 'Empty Rooms' (Fig. 55) to see whether the intrinsic reward can learn to encourage exploration of uncertain states like novelty-based exploration methods. The goal is to visit an invisible goal location, which is fixed within each lifetime but varies across lifetimes. An episode terminates when the goal is reached. Each lifetime consists of 200 episodes. From the agent's perspective, its policy should visit the locations suggested by the intrinsic reward. From the intrinsic reward's perspective, it should encourage the agent to go to unvisited locations to locate the goal, and then to exploit that knowledge for the rest of the lifetime.

Figure 56 shows that the learned intrinsic reward was more efficient than extrinsic rewards and count-based exploration when training a new agent. We observed that the intrinsic reward learned two interesting strategies as visualised in Figure 57. While the goal is not found, it encourages exploration of unvisited locations, because it learned the knowledge that there exists a rewarding goal location somewhere. Once the goal is found the intrinsic reward encourages the agent to exploit it without further exploration, because it learned that there is only one goal. This result shows that



(a) Room (b) Intrinsic (c) Count (d) ICM

Figure 57: Visualisation of the First 3000 Steps of an Agent Trained with Different Reward Functions in Empty Rooms. (a) The blue and yellow squares represent the agent and the *hidden* goal, respectively. (b) The learned reward encourages the agent to visit many locations if the goal is not found (top). However, when the goal is found early, the intrinsic reward makes the agent exploit it without further exploration (bottom). (c-d) Both the count-based and ICM rewards tend to encourage exploration (top) but hinders exploitation when the goal is found (bottom).

curiosity about uncertain states can naturally emerge when various states can be rewarding in a domain, even when the rewarding states are fixed within an agent's lifetime.

Exploring Uncertain Objects In the previous domain, we considered uncertainty of where the reward (or goal location) is. We now consider dealing with uncertainty about the value of different objects. In the 'Random ABC' environment (see Figure 55), for each lifetime the rewards for objects A, B, and C are uniformly sampled from $[-1, 1]$, $[-0.5, 0]$, and $[0, 0.5]$ respectively but are held fixed within the lifetime. A good intrinsic reward should learn that: 1) B should be avoided, 2) A and C have uncertain rewards, hence require systematic exploration (first go to one and then the other), and 3) once it is determined which of the two A or C is better, exploit that knowledge by encouraging the agent to repeatedly go to that object for the rest of the lifetime.

Figure 56 shows that the agent learned a near-optimal exploration-and-then-exploitation method with the learned intrinsic reward. Note that the agent cannot pass information about the reward for objects across episodes, as usual in reinforcement learning. The intrinsic reward can propagate such information across episodes and help the agent explore or exploit appropriately. We visualised the learned intrinsic reward for different actions sequences in Figure 58. The intrinsic rewards encourage the agent to explore towards A and C in the first few episodes. Once A and C are explored, the agent exploits the largest rewarding object. Throughout training, the agent is discouraged to visit B through negative intrinsic rewards. These results show that avoidance and curiosity about

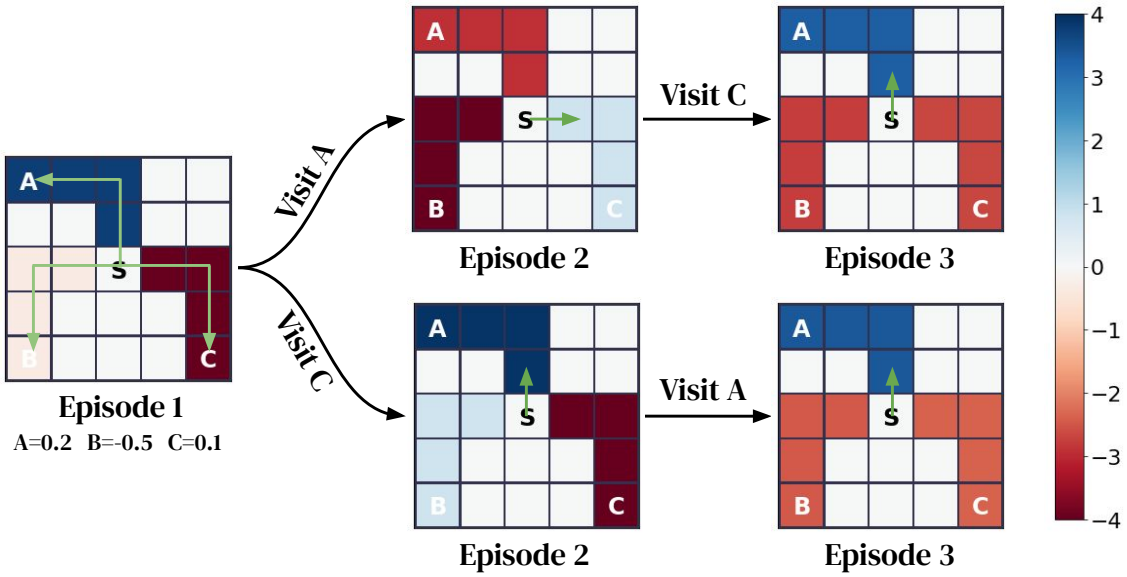


Figure 58: Visualisation of the Learned Intrinsic Reward in Random ABC, where the Extrinsic Rewards for A, B, and C are 0.2, -0.5, and 0.1 Respectively. Each figure shows the sum of intrinsic rewards for a trajectory towards each object (A, B, and C). In the first episode, the intrinsic reward encourages the agent to explore A. In the second episode, the intrinsic reward encourages exploring C if A is visited (top) or vice versa (bottom). In episode 3, after both A and C are explored, the intrinsic reward encourages revisiting A (both top and bottom).

uncertain objects can potentially emerge if the environment has various or fixed rewarding objects.

Exploiting Invariant Causal Relationship To see how the intrinsic reward deals with causal relationship between objects, we designed 'Key-Box', which is similar to Random ABC except that there is a key in the room (see Fig. 55). The agent needs to collect the key first to open one of the boxes (A, B, and C) and receive the corresponding reward. The rewards for the objects are sampled from the same distribution as Random ABC. The key itself gives a neutral reward of 0. Moreover, the locations of the agent, the key, and the boxes are randomly sampled for each episode. As a result, the state space contains more than 3 billion distinct states and thus is infeasible to fully enumerate. Figure 56 shows that learned intrinsic reward leads to a near-optimal exploration. The agent trained with extrinsic rewards did not learn to open any box. The intrinsic reward captures that the key is necessary to open any box, which is true across many lifetimes of training. This demonstrates that the intrinsic reward can capture causal relationships between objects when the domain has this kind of invariant dynamics.

Dealing with Non-stationarity We investigated how the intrinsic reward handles non-stationary tasks within a lifetime in our 'Non-stationary ABC' environment. Rewards are as follows: for A is either 1 or -1, for B is -0.5, for C is the negative value of the reward for A. The rewards of A and C are swapped every 250 episodes. Each lifetime lasts 1000 episodes. Figure 56 shows that the agent with the learned intrinsic reward quickly recovered its performance when the task changes, whereas the baselines take more time to recover. Figure 59 shows how the learned intrinsic reward encourages the learning agent to react to the changing rewards. Interestingly, the intrinsic reward has learned to prepare for the change by giving negative rewards to the exploitation policy

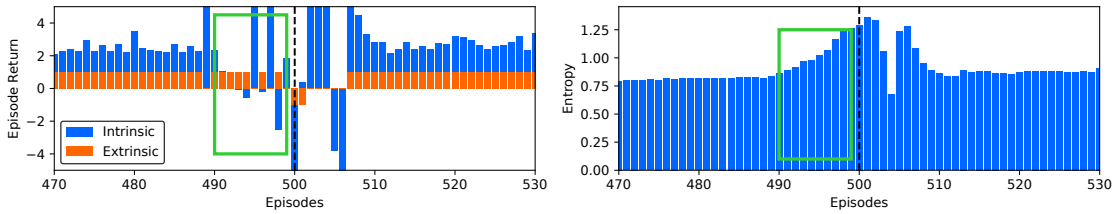


Figure 59: Visualisation of the agent's intrinsic and extrinsic rewards (left) and the entropy of its policy (right) on Non-stationary ABC. The task changes at 500th episode (dashed vertical line). The intrinsic reward gives a negative reward even before the task changes (green rectangle) and makes the policy less deterministic (entropy increases). As a result, the agent quickly adapts to the change.

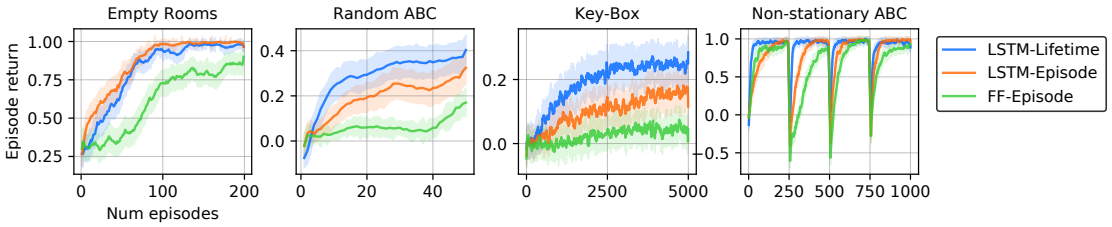


Figure 60: Evaluation of Different Intrinsic Reward Architectures and Objectives. For 'LSTM' the reward network has an LSTM taking the lifetime history as input. For 'FF' a feed-forward reward network takes only the current time-step. 'Lifetime' and 'Episode' means the lifetime and episodic return as objective respectively.

of the agent a few episodes before the task changes. In other words, the intrinsic reward reduces the agent's commitment to the current best rewarding object, thereby increasing entropy in the current policy in anticipation of the change, eventually making it easier to adapt quickly. This shows that the intrinsic reward can capture the (regularly) repeated non-stationarity across many lifetimes and make the agent intrinsically motivated not to commit too firmly to a policy, in anticipation of changes in the environment.

Ablation Study To study relative benefits of the proposed technical ideas, we conducted an ablation study 1) by replacing the long-term lifetime return objective (G^{life}) with the episodic return (G^{ep}) and 2) by restricting the input of the reward network to the current time-step instead of the entire lifetime history. Figure 60 shows that the lifetime history was crucial to achieve good performance. This is reasonable because all domains require some past information (e.g., object rewards in Random ABC, visited locations in Empty Rooms) to provide useful exploration strategies. It is also shown that the lifetime return objective was beneficial on Random ABC, Non-stationary ABC, and Key-Box. These domains require exploration across multiple episodes in order to find the optimal policy. For example, collecting an uncertain object (e.g., object A in Random ABC) is necessary even if the episode terminates with a negative reward. The episodic value function would directly penalise such an under-performing exploratory episode when computing meta-gradient, which prevents the intrinsic reward from learning to encourage exploration across episodes. On the other hand, such behaviour can be encouraged by the lifetime value function, as long as it provides

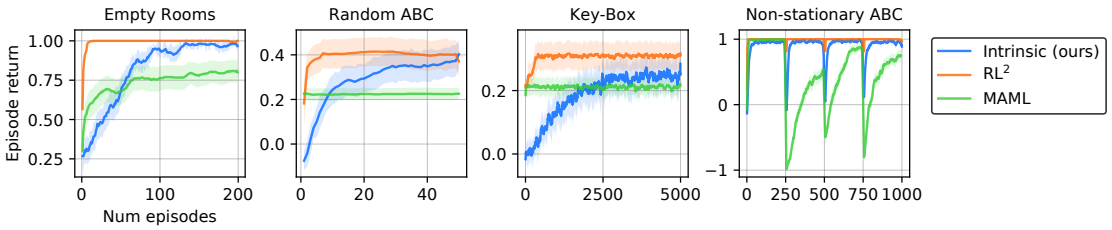
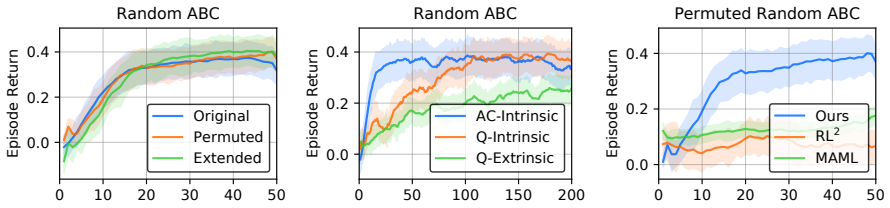


Figure 61: Comparison to Policy Transfer Methods.



(a) Action space **(b) Algorithm** **(c) Comparison to baselines**

Figure 62: Generalisation to New Agent-Environment Interfaces in Random ABC. (a) 'Permuted' agents have different action semantics. 'Extended' agents have additional actions. (b) 'AC-Intrinsic' is the original actor-critic agent trained with the intrinsic reward. 'Q-Intrinsic' is a Q-learning agent with the intrinsic reward learned from actor-critic agents. 'Q-Extrinsic' is the Q-learning agent with the extrinsic reward. (c) The performance of the policy transfer baselines with permuted actions during evaluation.

useful information to maximise the lifetime return in the long term.

4.4.6 Learning Curriculum Policies for Reinforcement Learning.

We evaluated learning curriculum policies for agents on a grid world domain [58] as well as a Ms. Pac-Man domain [195]. These domains were selected because they allow us to compare to previous methods; test our approach using different agent representations, different transfer learning algorithms, and different CMDP representations; and test its scalability to a more complex setting.

We will show the results as *CMDP learning curves*. The x-axis on these learning curves are over *CMDP episodes*. Each CMDP episode represents an execution of the current curriculum policy for the agent. Thus, multiple tasks are selected over the course of a single CMDP episode, with each task taking a varying number of steps/episodes, which contributes to the cost on the y-axis. Tasks are selected until the desired performance can be achieved in the target task, at which point the CMDP episode is terminated. In short, the curves show how long it would take to achieve a certain performance threshold on the target task following a curriculum, where the curriculum is represented by the CMDP policy, which is being learned over time.

We compare curriculum policies learned for each agent to two static curricula. The first is the baseline *no curriculum* policy. In this case, on each episode, the agent learns tabula rasa directly on the target task. The flat line plotted represents the average time needed to learn the target task directly. Note that the line is flat because the "curriculum" is fixed and does not change over time.

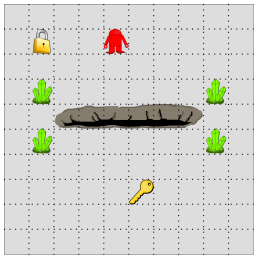


Figure 63: Grid World Target Task.

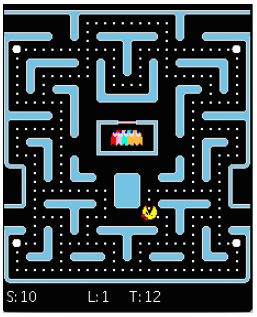


Figure 64: Ms. Pac-Man Target Task.

The second is a curriculum produced by following an existing curriculum algorithm (from [58] for the gridworld and from [59] for Ms. Pac-Man, to compare with past work). We also compare to a naive learning-based approach, which represents CMDP states using a list of all tasks learned by the learning agent. For example, the start state is the empty list. Upon learning a task M_1, the CMDP agent transitions to a new state $[M_1]$. If the CMDP agent subsequently selects task M_t, the resulting state is $[M_1, M_t]$. Note that this representation is a cruder approximation of the underlying process, as learning 2 different tasks that impart the same knowledge will lead to 2 different states under this representation. In order to deal with the combinatorial explosion of the size of the state space with this naive representation, we limit the number of tasks that can be used as sources in the curriculum to a constant (between 1 and 3 in our experiments), and force the selection of the target task after.

First we examine learning curriculum policies for 3 learning agents that have different state and action spaces [58], but use the same transfer learning algorithm (value function transfer), in a grid world domain (see Fig. 63). In addition, we compare the effect of two different types of representations for the CMDP state. The first CMDP representation is based on the finite state space representation discussed earlier, while the second CMDP representation is created directly from $\boldsymbol{\theta}$ without using an intermediary state-based action-value representation. In the next sections, we describe the domain and learning agents, followed by the representations for the CMDP state space and their effects on learning curriculum policies.

Gridworld Domain

Domain Description. The gridworld consists of a room with 4 different types of objects. *Keys* are objects the agent can pickup by executing a *pickup* action. These are used to unlock *locks*, which can depend on one or more keys, by executing an *unlock* action. *Pits* are obstacles that terminate the episode upon contact. Finally, *beacons* are landmarks placed on the 4 corners of a pit.

The goal of the agent is to traverse the world and unlock all the locks. To do so, at each step, the agent can move in one of the 4 cardinal directions, execute a pickup action, or an unlock action. Successfully picking up keys give a reward of +500, unlocking a lock rewards +1000, falling into a pit ends the episode with a reward of -200, and a constant step penalty of -10 is applied for all other actions.

Learning Agent Descriptions. We created 3 different learning agents that have varied sensing and action capabilities, based on the agents presented by Narvekar et. al. [58]. Creating these specific agents allows us to show that our approach works regardless of the state and action representation

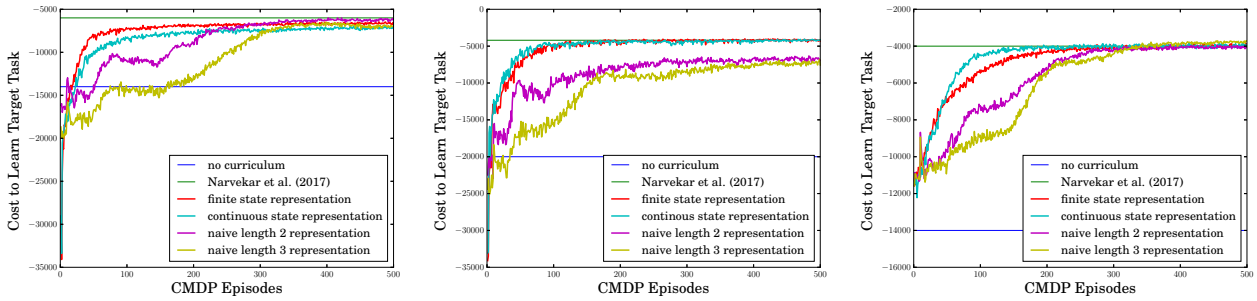


Figure 65: CMDP Learning Curves for the (a) Basic Agent, (b) Action-Dependent Agent, and (c) Rope Agent Using Different Curriculum Design Approaches and CMDP State Space Representations. The y-axis represents the cost (i.e., negative of the time needed) to reach a performance of 700 on the target task, following the curriculum policy at episode X. All curves are averaged over 500 runs. Each curriculum method was statistically significantly better than no curriculum using a 2 tail t-test with $p < 0.05$.

used by the learning agents, and also allows us to compare with the results of Narvekar et. al. [58]. We refer the reader there for full details of the agents, but recap the main elements and differences here for completeness.

The first agent, the *basic agent*, has 16 sensors, grouped into 4 on each side. The first sensor in each quadruple measures the Euclidean distance to the closest key from that side, the second the distance to the closest lock, the third the distance to the closest beacon, and the fourth detects whether there is a pit adjacent to the agent in that direction. An additional sensor indicates whether all keys in the room have been picked up, referred to as the *noKey* sensor. The agent used Sarsa(λ) with ϵ-greedy action selection, value function transfer for transfer learning, and CMAC tile coding with linear function approximation, with tile widths set to 1.

For the basic agent, we created two tilings: one over the 13 percepts from the key, beacon, pit, and noKey sensors, and another over the 13 percepts from the lock, beacon, pit, and noKey sensors. These tilings formed $\boldsymbol{\phi}(s)$ for the basic agent. The exploration rate ϵ was set to 0.1, eligibility trace parameter λ to 0.9, and learning rate α to 0.1 (these values match those reported in [58]).

The second, *action-dependent agent*, has the same sensors as the basic agent, but they are tiled differently, leading to a different $\boldsymbol{\phi}(s)$: one tiling is over the lock, pit, and noKey features; a second is over the key, pit, and noKey features; and a third is over the beacon and pit features. In addition, unlike the basic agent, the state representation is action-dependent. That is, when considering the *move right* action, the agent's feature vector uses values only from the right side sensors. The weights in the tilings are shared, so that the same set of weights is used for the state in each of the directions.

Finally, the *rope agent* is like the basic agent, except that it has 4 additional actions, which are to use a rope in one of the four directions. Doing so opens a path across a pit if one is present, and incurs the step cost of -10.

Table 14: Properties of Tasks in the Gridworld Experiments

Task Num	Grid Size	Num Keys	Num Locks	Pit Present	Rope Required
1	5x5	1	0	No	No
2	10x10	1	0	No	No
3	5x5	0	1	No	No
4	10x10	0	1	No	No
5	7x1	1	0	Yes	Yes
6	7x6	1	0	Yes	Yes
7	7x1	0	1	Yes	Yes
8	7x6	0	1	Yes	Yes
9	7x7	1	0	Yes	No
10	7x7	0	1	Yes	No
Target	10x10	1	1	Yes	No

CMDP Description. We defined our curriculum MDP as follows: **State space.** The start state S_0^C was derived from an untrained, uniformly initialized learning agent. The set of terminal states S_f^C were all states where the learning agent's policy allowed it to achieve a return of at least 700 on the target task. This performance threshold was the maximum that all the agents could achieve after training to convergence on the target task. Representations used for the CMDP state space are described in the next section. **Action space.** Source tasks were created using the TaskSimplification and OptionSubGoals heuristics [196]. These heuristics create source tasks by simplifying the domain, for example by reducing the size of the grid or the number of keys, locks, and pits, and by changing the goal of the task to be picking up keys. A total of 10 different tasks were created, and with the target task, these formed the action space $\mathcal{A}^C$ of the CMDP agent. The properties of these source tasks are summarized in Table 14 ("Rope required" indicates tasks where a pit blocks direct paths from the agent to the goal, necessitating a rope action. When a lock is not present, the episode terminates when all keys are picked up). **Transition function.** The (unknown) transition function is stochastic, describing how learning a task changes a learning agent's policy.

Reward function. The environment returns a reward $r^C(s^C, a^C)$ as the negative of the time needed to learn task a^C from state s^C. A task is considered learned once the policy ceases to change for 10 episodes. Time is measured using game steps.

Learning on the CMDP was done using Sarsa(λ) with $\epsilon = 0.001$, $\lambda = 0.9$, and $\alpha = 0.1$.

CMDP State Space Representations. One of the main challenges addressed in this research is identifying a representation for the CMDP state space that is both generalizable and compact enough to enable efficient learning of a curriculum for a range of agents. To this end, we instantiated and evaluated two forms for $\boldsymbol{\phi}^C$.

Recall that the learning agents use tile coding with linear function approximation. Here, $\boldsymbol{\phi}$ is a feature vector that indicates which tiles have been activated for state s and action a, and $\boldsymbol{\theta}$ are the corresponding weights in each tile. These weights $\boldsymbol{\theta}$ form the raw CMDP state variables s^C. We discuss two different ways to construct $\boldsymbol{\phi}^C(\boldsymbol{\theta})$, which will convert the raw state variables into a CMDP basis feature space suitable for learning.

Finite State Representation. The learning agents use Sarsa(λ) with an *egocentric* feature space. Thus, the parameters $\boldsymbol{\theta}$ learned are not action-values for each state. However, since the underlying domain has a fixed number of states, we can simulate the finite state representation case by moving

the learning agent to each of the states in the target task and computing action values. Let this new parameter of weights be $\boldsymbol{\theta}'$. We can now utilize the procedure described in Section 3.3.6 to create a CMDP feature space $\boldsymbol{\phi}^C(\boldsymbol{\theta}')$.

Continuous State Representation. The above representation is only well-defined in environments with a discrete underlying state space. We therefore also explore a CMDP representation that can apply in continuous domains by creating $\boldsymbol{\phi}^C$ directly from $\boldsymbol{\theta}$ without using an intermediary state-based action-value representation. Recall that the CMDP state variables $s^C = \boldsymbol{\theta}$ are the weights associated with all the tiles. Each of these tiles is part of a tiling group. For example, the basic and rope agents had 2 tiling groups over different subsets of its sensor percepts, while the action-dependent agent had 3 tiling groups. All tiles in a tiling group are related to each other. Thus there is an inherent structure to the parameters in the tiles.

However, forming a $\boldsymbol{\phi}^C$ tiling group over the weights of all the tiles associated with a $\boldsymbol{\phi}$ tiling would not generalize well, because it would require nearly identical action-preferences in every state to activate common tiles. Therefore, we created a separate tile group for each θ_i. Since the weights $\boldsymbol{\theta}$ within each learning agent's tilings $\boldsymbol{\phi}$ are still correlated, we normalized the weights associated within each $\boldsymbol{\phi}$ tile group.

Results and Discussion We learned curriculum policies for all 3 learning agents using both the finite and continuous state representations for the CMDP state space. The target task M_t is shown in Figure 63. The corresponding CMDP learning curves are shown in Figures 65(a) - 65(c). The results show that each agent successfully learned curriculum policies using both CMDP representations that were better than learning without a curriculum, and comparable to the curricula generated by previous work [58]. However, unlike this previous work, our approach does not require additional prior information about source tasks (such as task descriptors). In addition, the results show that our approach is robust to different predefined agent and CMDP representations.

So far, we demonstrated that CMDPs can be learned for agents with different actions and/or state representations. Another relevant way in which agents can differ is the algorithm by which they transfer knowledge from a source to a target task. Thus, in this section, we evaluate the robustness of our approach to different underlying transfer learning methods, while simultaneously evaluating the scalability to a significantly more complex Ms. Pac-Man domain (see Fig. 64). In particular, we examine the case when the learning agent stays the same, but uses 2 different types of transfer learning methods: value function transfer and reward shaping. The change in transfer algorithm affects both the CMDP state space representation, and the CMDP transition function, which we will describe in the following sections.

Ms. Pac-Man Experiments

Domain Description. Our implementation is based on code released by Taylor et al. [195] and augmented by Svetlik et al. [59]. The goal of the Ms. Pac-Man agent is to traverse a maze and accumulate points by eating edible objects such as food pellets, while avoiding ghosts. At each time step, Ms. Pac-Man can move along one of the 4 cardinal directions (though not every action is available in every state). The agent receives a reward of 10 for eating a pill and 50 for a power pill. Eating a power pill temporarily makes the ghosts edible. Eating the first ghost gives a reward of 200; each subsequent ghost eaten multiplies this reward by a factor of 2. The game ends when all food pellets and power pills have been eaten, a ghost eats Ms. Pac-Man, or when a time limit of 2000 game steps have occurred.

While the domain is technically discrete, it has a combinatorially large state space. There are over a thousand positions in the target task maze, and the state consists of the locations of Ms. Pac-Man, each food pellet, power pill, each of the 4 ghosts, the last move of each ghost, and whether each ghost is edible. Thus, it is essential for the Ms. Pac-Man learning agent to use function approximation.

Learning Agent Description We created a Ms. Pac-Man learning agent using the low-asymptote feature set described in Taylor et al. [195], Svetlik et al. [59]. The state space of the agent is represented by a set of action-dependent egocentric features, that count the fraction of pills, power pills, ghosts, and edible ghosts there are in each direction up to different "depths." The depth is represented in terms of junctions, i.e., locations in the maze where there are more than 2 possible actions. For example, the ghost feature for depth 1 would return the fraction of ghosts there are along one direction until the first junction. The features were used to learn a linear value function approximator.

The agent was trained using Sarsa(λ), with $\epsilon = 0.05$, $\alpha = 0.001$, $\gamma = 0.999$, and $\lambda = 0.9$. See the code by Svetlik et al.[59] for implementation details.

CMDP Description We defined our curriculum MDP as follows:

State space:

As before, the start state S_0^C was an untrained, randomly initialized learning agent. The set of terminal states S_f^C were all states where the learning agent could achieve a return of at least 2000 on the target task.

Action space:

We used the same 15 tasks used in the code release of et al. [59] to form the action space $\mathcal{A}^C$. These tasks were formed by varying the type of maze, as well as the number of pills, ghosts, and power pills. Their properties are summarized in Table 15. "Num Junctions" indicates how many maze positions had 3 or more direction actions possible. Note that some tasks have similar properties; however, the layout of the maps in these tasks differed. Please see the code release [59] for more details.

Transition function:

As before, the (unknown) transition function is stochastic, describing how Ms. Pac-Man's value function or set of shaping potentials changes as a result of learning a task.

Reward function:

We measure the cost of learning a task in terms of the number of game steps needed. Following the experimental setup of [59], a task is considered learned when at least 35% of the maximum reward possible for that task can be achieved. The maximum reward for a task is calculated analytically by summing the points accrued for eating all the pills, and all the edible ghosts for each power pill.

Learning on the CMDP was done using Sarsa(λ) with $\epsilon = 0.001$, $\lambda = 0.9$, and $\alpha = 0.05$.

CMDP State Space Representations. We consider 2 different CMDP state space representations that result from the use of 2 different transfer learning algorithms. In the value function transfer case, the raw CMDP state variables s^C are the weights $\boldsymbol{\theta}$ of the Ms. Pac-Man agent's linear function approximator. To create the CMDP space $\boldsymbol{\phi}^C$, we normalize $\boldsymbol{\theta}$ and use tile coding, creating a separate tiling over each θ_i. In the reward shaping setting, each source task in the curriculum is

Table 15: Properties of Source Tasks in the Ms. Pac-Man Experiments

Task Num	Num Junctions	Num Ghosts	Num Pills	Num Power Pills
1	2	0	53	1
2	2	1	65	2
3	40	2	234	4
4	36	4	240	4
5	8	0	179	4
6	8	2	179	4
7	8	4	179	4
8	13	2	209	4
9	13	4	209	4
10	13	0	209	4
11	24	0	231	4
12	24	2	231	4
13	24	4	231	4
14	24	4	231	4
Target	36	4	240	4

associated with a potential function (derived from the value function). As multiple tasks are learned, the potentials are added together, and used to create a shaping reward (as done in Svetlik et al.[59]). Thus, the raw CMDP state variables are the summed weights of the potential functions. As in the value function case, we use tile coding to create a separate tiling over each potential weight feature to create the CMDP basis space.

Results and Discussion. Figure 66(a) shows CMDP learning curves for Ms. Pac-Man using value function transfer and Figure 66(b) shows the curves using transfer with reward shaping. Each curriculum method was statistically significantly better than no curriculum. In (b), the CMDP-based approaches were statistically better than et al.[59]. In (c), the "small fixed" approaches were statistically better than their corresponding "return-based" methods. These were tested using a 2-tail t-test with $p < 0.05$. The results again clearly demonstrate that curriculum policies can be learned, and that such policies are more useful than training directly on the target task. They also show that the approach is adjustable to different types of transfer learning algorithms. In addition, we compared the reward shaping approach with that of Svetlik et al.[59], who also use reward shaping for transfer in their curriculum algorithm, and found that a much better curriculum is possible in this more complex domain.[9]

Finally, we also study the effect of the hyperparameter that controls when to finish training on a source task. For the previous two experiments in Ms. Pac-Man, training on a source was stopped after 35% of the max possible return in the task was achieved, to replicate the experimental conditions of et al.[59]. Since their approach precomputes a curriculum and does not model the state of the learning agent's progress, this termination condition must be carefully chosen to ensure something can be learned in each source task. In contrast, with our approach, we can train on source tasks for an arbitrarily small amount of time, as the curriculum policy can learn to reselect a task if additional experience in that task is required.

In Figure 66(c), we reproduce the continuous state representation CMDP learning curves using

[9]Our results are based on a reproduction of their experiments using their publicly released code. Interestingly, we also get slightly better results for their method than they report in their paper. We measure cost in episodes for this experiment only to facilitate comparison to their work.

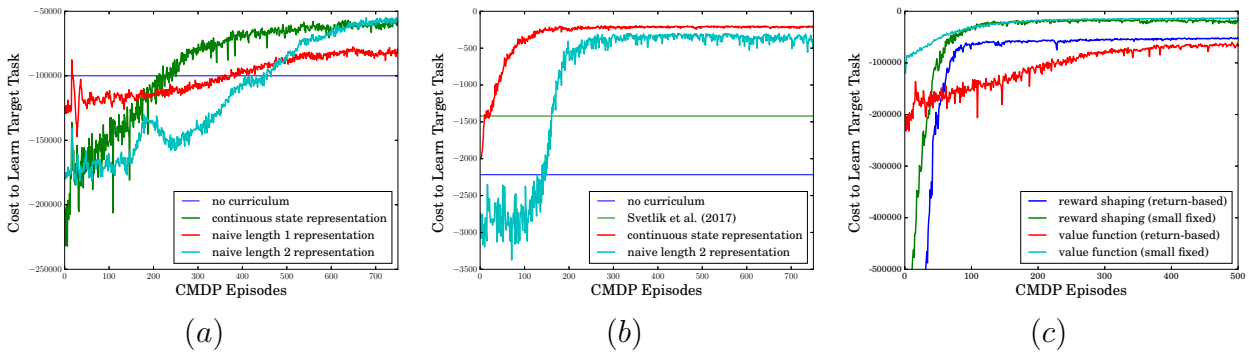


Figure 66: CMDP Learning Curves on the Ms. Pac-Man Target Task, using (a) Value Function Transfer, (b) Transfer with Reward Shaping, and (c) a Comparison Between the Continuous Representations for Value Function and Reward Shaping Transfer, using different criteria to determine when to stop training on source tasks. All curves are averaged over 500 runs. Cost is measured in game steps for (a) & (c), and episodes for (b).

value function transfer from Figure 66(a) and reward shaping from Figure 66(b). These are denoted in the figure by "(return-based)", and train on sources until 35% of the max return is achieved. We compare them against an approach that is identical to "(return-based)" approaches, but that trains for 5 episodes on a task at a time. These CMDP learning curves are denoted with "(small fixed)." The results show that agents do not need to train for a long time or to convergence on source tasks, and that our approach can adapt to this hyperparameter setting.

Curriculum Generalization

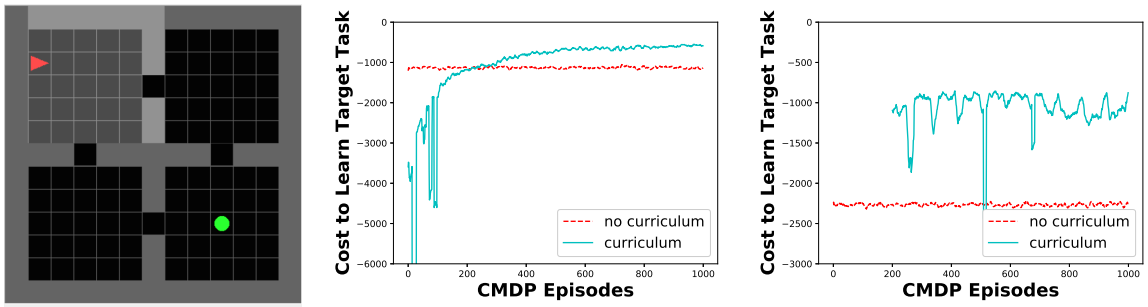
Domain description. We evaluated curriculum transfer on navigation tasks in a static gridworld environment. Our goal was to train a CMDP teacher agent to learn a curriculum policy on a subset of tasks, and show that it can zero-shot produce curricula for a student on novel unseen target tasks. The gridworld environment considers goal-oriented navigation tasks in a standard 4-room grid world. The environment consists of 4 connected rooms, where each room is 5x5 in size and connected at one cell to each adjacent room. A navigation task is defined by a pair of (x, y) coordinates for the starting position and goal position. There are 100 possible starting and ending positions. We ignore tasks that start and end on the same position, thus there are 9900 possible different target tasks. See Figure 67(a) for an example of a task.

Our student agent has a tabular representation for the state space, and learns using Sarsa(λ) with exploration $\epsilon = 0.1$, learning rate $\alpha = 0.1$, discount factor $\gamma = 1.0$, and eligibility trace decay rate $\lambda = 0.7$. We use value function transfer to transfer information between tasks in the curriculum.

Teacher (CMDP) Agent Description

State and goal space:

The CMDP needs to learn to generalize over both the agent's knowledge and task space. Conceptually, the agent's current policy – its function for selecting actions in each state, which we assume to be known to the teacher – is its state of knowledge. We thus represent the agent's knowledge using



(a) (b) (c)

Figure 67: Example CMDP Task and Learning Curves. (a) An example of a task in the gridworld environment. The red arrow is the agent, and the green circle is the goal. (b) CMDP learning curves for the interpolation experiments. (c) CMDP learning curves for the extrapolation experiments. In the CMDP learning curves, the x-axis represents a CMDP episode, where each episode is an entire run of a curriculum. The y-axis is the cost of that curriculum in game steps.

the vector of weights associated with the student's Q-function table. Tasks are represented using the pair of (x, y) coordinates associated with the starting and goal states.

Action space:

We create nine different source tasks. Eight of these are static tasks that don't change based on the target task. These tasks initialize the agent in one of the 4 rooms, and terminate when the agent moves into one of the adjacent two rooms. There is one such task for each room and adjacent room pair. In addition, all corridors between rooms are blocked except for the one required to complete the task. The ninth source task is a dynamic source task that changes based on the current target task. This task initializes the agent in the same room as the goal of the target task, and sets the goal tile to be the same as the target task. As with the static sources, all corridors to other rooms are blocked off. These sources, together with the target task, form the action space of the CMDP. Note that these tasks were intentionally designed to give rise to a natural and interpretable curriculum for each target task: use the static sources to navigate to the goal room, and follow with the dynamic source task to complete the path.

Reward function:

When a task is selected, it is trained on until convergence. We consider a task converged when the steps taken to reach the goal averaged over the last 5 episodes is less than the Manhattan distance between the start $\mathbf{s}$ and goal positions $\mathbf{g}$ plus a slack term δ:

$$\text{converged} := \big(\text{steps taken} < ||\mathbf{s} - \mathbf{g}||_1 + \delta\big) \tag{76}$$

We set δ to 0 when the start and goal positions are in the same room, 5 when they are in adjacent rooms, and 10 when they are diagonally across, to account for navigating around walls. The cost of learning a task is the number of steps needed to learn to convergence. Therefore, the reward given is the negative of the steps taken.

Architecture and learning parameters. We use DQN [197] to learn the CMDP. The neural network uses a two-stream architecture, where the features relating to the task/goal space pass through a single hidden layer, while the agent knowledge features pass through three hidden layers. Each hidden layer has 128 units followed by a $\tanh$ activation function. The two streams are subsequently merged via element-wise multiplication, and pass through a final hidden layer to produce action-values. A diagram of this network can be seen in Figure 9. The learning rate is set to 5e-4, replay buffer size to 5000, batch size 64, exploration fraction to 0.05, and has the target network updated every 50 steps. To speed up training, we also capped the number of actions the CMDP could take to 5, and trained on the target task thereafter. The learning parameters were not extensively optimized.

Results We consider two types of generalization that may be possible in CMDPs: interpolation and extrapolation. In the interpolation case, we randomly shuffle all the possible tasks, and present them to the CMDP agent one by one. This is the lifelong learning setting, where each task encountered is new. The results are shown in Figure 67(b). As the CMDP learns and the coverage of tasks increase, the curricula produced gradually improve, until they pass the baseline of just training on the target task after having seen 300 tasks. Thus, the results show how many tasks a curriculum must be learned for before the teacher agent is able to zero-shot produce curricula for novel unseen tasks.

In the extrapolation case, we explicitly split the set of target tasks into a train set and a test set. The test set contains all tasks that start in the top left room, and end in the bottom right room, while the train set contains tasks with all other possible start and end goal pairs. The extrapolation case is more challenging, because in the previous interpolation setting, generalization was expected because goals were represented with similar features and the set of training tasks covered pairings from all the different rooms. However, in this case, the curriculum of navigating from the top left room to the bottom right room has not been seen before. We train on tasks in the train set for the first 200 CMDP episodes, and subsequently evaluate on the test set. The results are shown in Figure 67(c), with the curriculum graph offset to reflect time spent training on the train set. The results once again show that curricula learned on a different set of tasks can transfer to produce curricula for new unseen tasks.

4.5 Evaluation of Other Lifelong Learning Algorithms and Approaches

In this section, we describe the evaluation of other lifelong learning algorithms developed under this project. The technical description of the corresponding algorithms are given in Section 3.4.

4.5.1 Compositional Lifelong Classification.

Framework instantiations We evaluated our framework with the three compositional structures of Section 3.4.1. All methods assimilate task $\mathcal{Z}^{(t)}$ via backpropagation on the structure's parameters $\boldsymbol{\psi}^{(t)}$. For each, we trained three instantiations of Algorithm 1, varying the method used for adaptation:

- Naïve fine-tuning (**NFT**) updates components via standard backpropagation, ignoring past tasks.

- Elastic weight consolidation (**EWC**, [73]) uses $\frac{\lambda}{2}\sum_{t=1}^{T-1}\|\boldsymbol{\theta}-\boldsymbol{\theta}^{(t)}\|^2_{\boldsymbol{F}^{(t)}}$ to penalize modifying model parameters, where $\boldsymbol{F}^{(t)}$ is the Fisher information around $\boldsymbol{\theta}^{(t)}$. Backpropagation is carried out on the regularized loss, and we approximated $\boldsymbol{F}^{(t)}$ with Kronecker factors.

- Experience replay (**ER**) stores n_m samples per task in a replay buffer, and during adaptation takes backpropagation steps with data from the replay buffer along with the current task's data.

We explored variations with and without the expansion step: **dynamic + compositional** methods use component dropout to add new components, while **compositional** methods keep a fixed-size set.

Baselines For every adaptation method listed above, we constructed two baselines. **Joint** baselines use compositional structures, but do not separate assimilation and accommodation, and instead update components and structures jointly. In contrast, **no-components** baselines optimize a single architecture to be used for all tasks, with additional task-specific input and output mappings, $\mathcal{E}^{(t)}$ and $\mathcal{D}^{(t)}$. The latter baselines correspond to the most common lifelong learning approach, which learns a monolithic structure shared across tasks, while the former are the naïve extensions of those methods to a compositional setting. We also trained an ablated version of our framework that keeps all components fixed after initialization (**FM**), only taking assimilation steps for each new task.

Linear combinations of models We first evaluated linear combinations of models on three data sets used previously for evaluating linear lifelong learning [35]. The Facial Recognition (**FERA**) data set tasks involve recognizing one of three facial expression action units for one of seven people, for a total of $T = 21$ tasks. The **Landmine** data set consists of $T = 29$ tasks, which require detecting land mines in radar images from different regions. Finally, the London Schools (**Schools**) data set contains $T = 139$ regression tasks, each corresponding to exam score prediction in a different school.

Table 16 summarizes the results obtained with linear models. The compositional versions of ER, EWC, and NFT clearly outperformed all the joint versions, which learn the same form of models but by jointly optimizing structures and components. This suggests that the separation of the learning process into assimilation and accommodation stages enables the agent to better capture the structure of the problem. Interestingly, the no-components variants, which learn a single linear model for all tasks, performed better than the jointly trained versions in two out of the three data sets, and even outperformed our compositional algorithms in one. This indicates that the tasks in those two data sets (Landmine and Schools) are so closely related that a single model can capture them.

Deep compositional learning with soft layer ordering We then evaluated how the different algorithms performed when learning deep nets with soft layer ordering, using five data sets. Binary MNIST (**MNIST**) is a common lifelong learning benchmark, where each task is a binary classification problem between a pair of digits. We constructed $T = 10$ tasks by randomly sampling the digits with replacement across tasks. The Binary Fashion MNIST (**Fashion**) data set is similar to MNIST, but images correspond to items of clothing. For these two data sets, all models used a task-specific input transformation layer $\mathcal{E}^{(t)}$ initialized at random and kept fixed throughout training, to ensure that the input spaces were sufficiently different [72]. A more complex lifelong learning problem commonly used in the literature is Split CUB-200 (**CUB**), where the agent must classify bird

Table 16: Average Final Performance Across Tasks Using Factored Linear Models—Accuracy for FERA and Landmine and RMSE for Schools

Base	Algorithm	FERA	Landmine	Schools
ER	Compositional	$\mathbf{79.0 \pm 0.4}\%$	$\mathbf{93.6 \pm 0.1}\%$	10.65 ± 0.04
	Joint	$78.2 \pm 0.4\%$	$90.5 \pm 0.3\%$	11.55 ± 0.09
	No Comp.	$66.4 \pm 0.3\%$	$93.5 \pm 0.1\%$	$\mathbf{10.34 \pm 0.02}$
EWC	Compositional	$\mathbf{79.0 \pm 0.4}\%$	$\mathbf{93.7 \pm 0.1}\%$	10.55 ± 0.03
	Joint	$72.1 \pm 0.7\%$	$92.2 \pm 0.2\%$	10.73 ± 0.17
	No Comp.	$60.1 \pm 0.5\%$	$93.5 \pm 0.1\%$	$\mathbf{10.35 \pm 0.02}$
NFT	Compositional	$\mathbf{79.0 \pm 0.4}\%$	$\mathbf{93.7 \pm 0.1}\%$	$\mathbf{10.87 \pm 0.07}$
	Joint	$67.9 \pm 0.6\%$	$72.8 \pm 2.5\%$	25.80 ± 2.35
	No Comp.	$57.0 \pm 0.9\%$	$92.7 \pm 0.4\%$	18.01 ± 1.04

Table 17: Average Final Accuracy Across Tasks Using Soft Layer Ordering

Base	Algorithm	MNIST	Fashion	CUB	CIFAR	Omniglot
ER	Dyn. + Comp.	$\mathbf{97.6 \pm 0.2}\%$	$\mathbf{96.6 \pm 0.4}\%$	$79.0 \pm 0.5\%$	$\mathbf{77.6 \pm 0.3}\%$	$\mathbf{71.7 \pm 0.5}\%$
	Compositional	$96.5 \pm 0.2\%$	$95.9 \pm 0.6\%$	$\mathbf{80.6 \pm 0.3}\%$	$58.7 \pm 0.5\%$	$71.2 \pm 1.0\%$
	Joint	$94.2 \pm 0.3\%$	$95.1 \pm 0.7\%$	$77.7 \pm 0.5\%$	$65.8 \pm 0.4\%$	$70.7 \pm 0.3\%$
	No Comp.	$91.2 \pm 0.3\%$	$93.6 \pm 0.6\%$	$44.0 \pm 0.9\%$	$51.6 \pm 0.6\%$	$43.2 \pm 4.2\%$
EWC	Dyn. + Comp.	$\mathbf{97.2 \pm 0.2}\%$	$\mathbf{96.5 \pm 0.4}\%$	$\mathbf{73.9 \pm 1.0}\%$	$\mathbf{77.6 \pm 0.3}\%$	$\mathbf{71.5 \pm 0.5}\%$
	Compositional	$96.7 \pm 0.2\%$	$95.9 \pm 0.6\%$	$73.6 \pm 0.9\%$	$48.0 \pm 1.7\%$	$53.4 \pm 5.2\%$
	Joint	$66.4 \pm 1.4\%$	$69.6 \pm 1.6\%$	$65.4 \pm 0.9\%$	$42.9 \pm 0.4\%$	$58.6 \pm 1.1\%$
	No Comp.	$66.0 \pm 1.1\%$	$68.8 \pm 1.1\%$	$50.6 \pm 1.2\%$	$36.0 \pm 0.7\%$	$68.8 \pm 0.4\%$
NFT	Dyn. + Comp.	$\mathbf{97.3 \pm 0.2}\%$	$\mathbf{96.4 \pm 0.4}\%$	$73.0 \pm 0.7\%$	$\mathbf{73.0 \pm 0.4}\%$	$\mathbf{69.4 \pm 0.4}\%$
	Compositional	$96.5 \pm 0.2\%$	$95.9 \pm 0.6\%$	$\mathbf{74.5 \pm 0.7}\%$	$54.8 \pm 1.2\%$	$68.9 \pm 0.9\%$
	Joint	$67.4 \pm 1.4\%$	$69.2 \pm 1.9\%$	$65.1 \pm 0.7\%$	$43.9 \pm 0.6\%$	$63.1 \pm 0.9\%$
	No Comp.	$64.4 \pm 1.1\%$	$67.0 \pm 1.3\%$	$49.1 \pm 1.6\%$	$36.6 \pm 0.6\%$	$68.9 \pm 1.0\%$
FM	Dyn. + Comp.	$\mathbf{99.1 \pm 0.0}\%$	$\mathbf{97.3 \pm 0.3}\%$	$78.3 \pm 0.4\%$	$\mathbf{78.4 \pm 0.3}\%$	$\mathbf{71.0 \pm 0.4}\%$
	Compositional	$84.1 \pm 0.8\%$	$86.3 \pm 1.3\%$	$\mathbf{80.1 \pm 0.3}\%$	$48.8 \pm 1.6\%$	$63.0 \pm 3.3\%$

species. We created $T = 20$ tasks by randomly sampling ten species for each, without replacement across tasks. All agents used a frozen ResNet-18 pre-trained on ImageNet as a feature extractor $\mathcal{E}^{(t)}$ shared across all tasks. For these first three data sets, all architectures were fully connected networks. To show that our framework supports more complex convolutional architectures, we used two additional data sets. We constructed a lifelong learning version of CIFAR-100 (**CIFAR**) with $T = 20$ tasks by randomly sampling five classes per task, without replacement across tasks. Finally, we used the **Omniglot** data set, which consists of $T = 50$ multi-class classification problems, each corresponding to detecting handwritten symbols in a given alphabet. The inputs to all architectures for these two data sets were the images directly, without any transformation $\mathcal{E}^{(t)}$.

Results in Table 17 show that all the algorithms conforming to our framework outperformed the joint and no-components learners. In four out of the five data sets, the dynamic addition of new components yielded either no or marginal improvements. However, on CIFAR it was crucial for

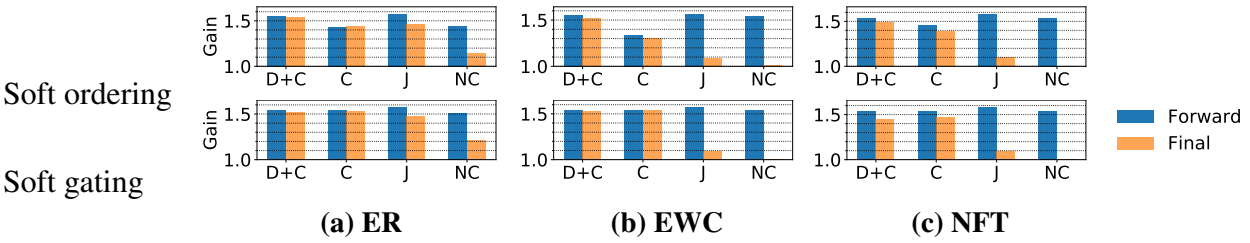
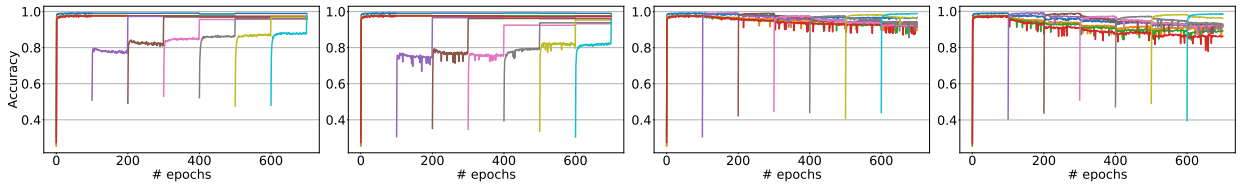


Figure 68: Average Gain W.R.T. No-Components NFT Across Tasks and Data Sets Immediately After Training on Each Task (Forward) and After All Tasks Had Been Trained (Final), Using Soft Ordering (Top) and Soft Gating (Bottom)

the agent to be capable of detecting when new components were needed. This added flexibility enables our learners to handle more varied tasks, where new problems may not be solved without substantially new knowledge. Algorithms with adaptation outperformed the ablated compositional FM agent, showing that it is necessary to accommodate new knowledge into the set of components in order to handle a diversity of tasks. When FM was allowed to dynamically add new components (keeping old ones fixed), it yielded the best performance on MNIST and Fashion by adding far more components than methods with adaptation, as well as on CIFAR.

To study how flexibly our agents learn new tasks and how stably they retain knowledge about earlier tasks, Figure 68 (top) shows accuracy gains immediately after each task was learned (forward) and after all tasks had been learned (final), w.r.t. no-components NFT (final). Compositional learners with no dynamically added components struggled to match the forward performance of joint baselines, indicating that learning the ordering over existing layers during much of training is less flexible than modifying the layers themselves, as expected. However, the added stability dramatically decreased forgetting w.r.t. joint methods. The dynamic addition of new layers yielded substantial improvements in the forward stage, while still reducing catastrophic forgetting w.r.t. baselines. Figure 69 shows the learning curves of MNIST and Fashion tasks using ER, the best adaptation method. Performance jumps in 100-epoch intervals show adaptation steps incorporating knowledge about the current task into the existing components without noticeably impacting earlier tasks' performance. Compositional and dynamic + compositional ER exhibit almost no performance drop after training on a task, whereas accuracy for the joint and no-components versions diminishes as the agent learns subsequent tasks. Most notably, as more tasks were seen by dynamic ER, the existing components became better able to assimilate new tasks, shown by the trend of increasing performance as the number of tasks increases. This suggests that the later tasks' accommodation stage can successfully determine which new knowledge should be incorporated into existing components (enabling them to better generalize across tasks), and which must be incorporated into a new component.

Deep compositional learning with soft gating Finally, we tested our algorithms when the compositional structures were given by a soft gating net. Table 18 shows a substantial improvement of compositional algorithms w.r.t. baselines. We hypothesized that the gating net granted our assimilation step more flexibility, which is confirmed in Figure 68 (bottom): the forward accuracy of compositional methods was nearly identical to that of jointly trained and no-components versions. This added flexibility enabled our simplest version of a compositional algorithm, FM, to perform



(a) ER Dyn. + Comp. **(b) ER Compositional** **(c) ER Joint** **(d) ER No Components**

Figure 69: Learning Curves Averaged Across MNIST and Fashion Using ER and Soft Ordering

better than the full versions of our algorithm on the CIFAR data set with convolutional gating nets, showing that even the components initialized with only a few tasks are sufficient for top lifelong learning performance. We attempted to run experiments with this method on the CUB data set, but found that all algorithms were incapable of generalizing to test data. This is consistent with findings in prior work, which showed that gating nets require vast amounts of data, unavailable in CUB [65, 71].

Deep compositional learning of sequences of diverse tasks One of the key advantages of learning compositional structures is that they enable learning a more diverse set of tasks, by

Table 18: Average Final Accuracy Across All Tasks Using Soft Gating

Base	Algorithm	MNIST	Fashion	CIFAR	Omniglot
ER	Dyn. + Comp.	$\mathbf{98.2 \pm 0.1\%}$	$\mathbf{97.1 \pm 0.4\%}$	$74.9 \pm 0.3\%$	$73.7 \pm 0.3\%$
	Compositional	$98.0 \pm 0.2\%$	$97.0 \pm 0.4\%$	$\mathbf{75.9 \pm 0.4\%}$	$\mathbf{73.9 \pm 0.3\%}$
	Joint	$93.8 \pm 0.3\%$	$94.6 \pm 0.7\%$	$72.0 \pm 0.4\%$	$72.6 \pm 0.2\%$
	No Comp.	$91.2 \pm 0.3\%$	$93.6 \pm 0.6\%$	$51.6 \pm 0.6\%$	$43.2 \pm 4.2\%$
EWC	Dyn. + Comp.	$\mathbf{98.2 \pm 0.1\%}$	$\mathbf{97.0 \pm 0.4\%}$	$76.6 \pm 0.5\%$	$73.6 \pm 0.4\%$
	Compositional	$98.0 \pm 0.2\%$	$\mathbf{97.0 \pm 0.4\%}$	$\mathbf{76.9 \pm 0.3\%}$	$\mathbf{74.6 \pm 0.2\%}$
	Joint	$68.6 \pm 0.9\%$	$69.5 \pm 1.8\%$	$49.9 \pm 1.1\%$	$63.5 \pm 1.2\%$
	No Comp.	$66.0 \pm 1.1\%$	$68.8 \pm 1.1\%$	$36.0 \pm 0.7\%$	$68.8 \pm 0.4\%$
NFT	Dyn. + Comp.	$\mathbf{98.2 \pm 0.1\%}$	$\mathbf{97.1 \pm 0.4\%}$	$66.6 \pm 0.7\%$	$69.1 \pm 0.9\%$
	Compositional	$98.0 \pm 0.2\%$	$96.9 \pm 0.5\%$	$\mathbf{68.2 \pm 0.5\%}$	$\mathbf{72.1 \pm 0.3\%}$
	Joint	$67.3 \pm 1.7\%$	$66.4 \pm 1.9\%$	$51.0 \pm 0.8\%$	$65.8 \pm 1.3\%$
	No Comp.	$64.4 \pm 1.1\%$	$67.0 \pm 1.3\%$	$36.6 \pm 0.6\%$	$68.9 \pm 1.0\%$
FM	Dyn. + Comp.	$\mathbf{98.4 \pm 0.1\%}$	$\mathbf{97.0 \pm 0.4\%}$	$\mathbf{77.2 \pm 0.3\%}$	$74.0 \pm 0.4\%$
	Compositional	$94.8 \pm 0.4\%$	$96.3 \pm 0.4\%$	$\mathbf{77.2 \pm 0.3\%}$	$\mathbf{74.1 \pm 0.3\%}$

Table 19: Average Final Accuracy Across Tasks on the Combined Data Set

Base	Algorithm	All data sets	MNIST	Fashion	CUB
ER	Dyn. + Comp.	$\mathbf{86.5 \pm 1.8\%}$	$\mathbf{99.5 \pm 0.0\%}$	$\mathbf{98.0 \pm 0.3\%}$	$\mathbf{74.2 \pm 2.0\%}$
	Compositional	$82.1 \pm 2.5\%$	$\mathbf{99.5 \pm 0.0\%}$	$97.8 \pm 0.3\%$	$65.5 \pm 2.4\%$
	Joint	$72.8 \pm 4.1\%$	$98.9 \pm 0.3\%$	$97.0 \pm 0.7\%$	$47.6 \pm 6.2\%$
	No Comp.	$47.4 \pm 4.5\%$	$91.8 \pm 1.3\%$	$83.5 \pm 2.5\%$	$7.1 \pm 0.4\%$

recombining components in novel ways to solve each problem. In this setting, non-compositional structures struggle to capture the diversity of the tasks in a single monolithic architecture. To verify that this is indeed the case, we created a novel data set that combines the MNIST, Fashion, and CUB data sets into a single **Combined** data set of $T = 40$ tasks. We trained all our instantiations and baselines with soft layer ordering, using the same architecture as used for CUB in the separate evaluation per data set. Agents were given no indication that each task came from a different data set. Table 19 summarizes the results for ER-based algorithms, showing that our method greatly outperforms the baselines. In particular, ER with no components was completely incapable of learning the CUB tasks, showing that compositional architectures are required to handle this more complex setting. Even jointly training the components and structures performed poorly. Algorithms under our framework instead performed remarkably well, especially the complete version with dynamically added components.

Visualization of the learned components We now visually inspect the components learned by our framework to verify that they are indeed self-contained and reusable. Similar to [72], we created $T = 10$ generative tasks, where each pixel in an image of the digit "4" constitutes one data point, using the coordinates as features and the pixel intensity as the label. We trained a soft ordering net with $k = 4$ components via compositional ER, and shared the input and output transformations across tasks to ensure the only differences across task models are due to the structure $s^{(t)}$ of each task. Varying the intensity $\boldsymbol{\psi}_{i,j}^{(t)}$ with which component i is selected at various depths j gives information about the effect of the component in different contexts. Figure 70 shows generated digits as the intensity of component $i = 0$ varies at different depths, revealing that the discovered component learned to vary the thickness of the digit regardless of the task at hand, with a more extreme effect at the initial layers.

4.5.2 Compositional Lifelong Reinforcement Learning.

We now define two domains that intuitively exhibit the properties outlined in Section 3.4.12.

Discrete 2-D world The agent's goal is to reach a specific target in an environment populated with static objects that have different effects. The tasks are compositional in three hierarchical levels:

- *Agent dynamics*: We artificially create four different dynamics, each corresponding to a permutation of the actions (e.g., the `turn_left` action moves the agent forward, `turn_right` rotates left). For each task, the effect of the agent's actions is determined by the chosen dynamics.

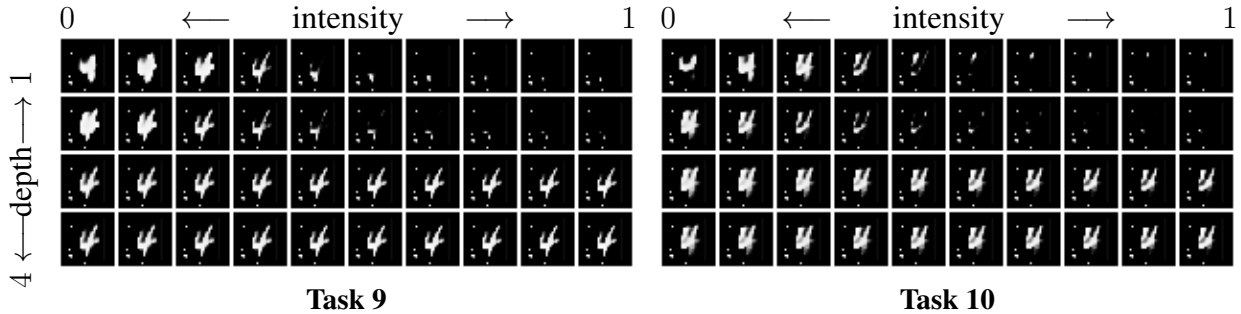


Figure 70: Generated MNIST "4" Digits on the Last Two Tasks Seen by Compositional ER With Soft Ordering, Varying the Intensity With Which a Specific Component Is Selected

- *Static objects*: We introduce a chain of static objects in each task, with a single gap cell. `Walls` block the agent's movement; the agent must open a `door` in the gap cell to move to the other side. `Floor` cells have an object indicator, but have no effect on the agent. `Food` gives a small positive reward if picked up. Finally, `lava` gives a small negative reward and terminates the episode.
- *Target objects*: There are four colors of targets; each task involves reaching a specific color.

There are 64 tasks of the form "reach the `COLOR` target with dynamics `N` interacting with `OBJECT`." If the agent has learned to "reach the red target with dynamics 0" and "reach the green target with dynamics 1", then it should be able to "reach the red target with dynamics 1" by recombining its knowledge. Tasks were simulated on `gym-minigrid` [170].

These 2-D tasks capture the notion of functional composition we study in this work, and prove to be notoriously difficult for existing lifelong RL methods. This demonstrates both the difficulty of the problem of knowledge composition and the plausibility of neural composition as a solution. To show the real-world applicability of the problem, we propose a second domain of different robot arms performing a variety of manipulation tasks that vary in three hierarchical levels.

Robot manipulation We consider four popular commercial robotic arms with seven degrees of freedom (7-DoF): Rethink Robotics' Sawyer, KUKA's IIWA, Kinova's Gen3, and Franka's Panda.

- *Robot dynamics*: These arms have varying kinematic configurations, joint limits, and torque limits, requiring specialized policies. All dynamics are simulated in `robosuite` [198].
- *Objects*: The robot must grasp and lift a can, a milk carton, a cereal box, or a loaf of bread. The varying sizes and shapes imply that no common strategy can manipulate all these objects.
- *Obstacles*: The workspace the robot must act in may be free (i.e., no obstacle), blocked by a wall the robot needs to circumvent, or limited by a door frame that the robot must traverse.

Each task is one of the 48 combinations of the above elements, just like in the 2-D case. Intuitively, if the agent has learned to manipulate the milk carton with the IIWA arm and the cereal box with the Panda arm, then it could recombine knowledge to manipulate the milk carton with the Panda arm.

Zero-shot transfer to unseen discrete 2-D task combinations via MTL To assess the compositionality of the tasks we constructed, we provided the agent with a fixed graph structure following the formalism of Section 3.4.12. This way, the agent knows *a priori* which modules to use for each task and need only learn the module parameters. The architecture uses four modules of each of three types, one for each task component (static object, target object, and agent dynamics). Each task policy contains one module of each type, chained as $\text{static object} \rightarrow \text{target object} \rightarrow \text{agent}$. Modules are convolutional nets whose inputs are the module-specific states and the outputs of the previous modules. Agent modules output both the action and the Q-values. We trained the agent in a multi-task fashion using PPO, collecting data from all tasks at each training step and computing the average gradient across tasks to update the parameters. Since each task uses a single module of each type, gradient updates only affect the relevant modules to each task.

We trained the agent on various sets of discrete 2-D tasks and compared it against two baselines: training a separate single-task (STL) agent on each task, and training a single monolithic network across all tasks in the same multi-task fashion. To ensure a fair comparison, the monolithic MTL network received as input a multi-hot encoding of the components that constituted each task. Figure 71 (left) shows that our method is substantially faster than the baselines by sharing relevant information across tasks.

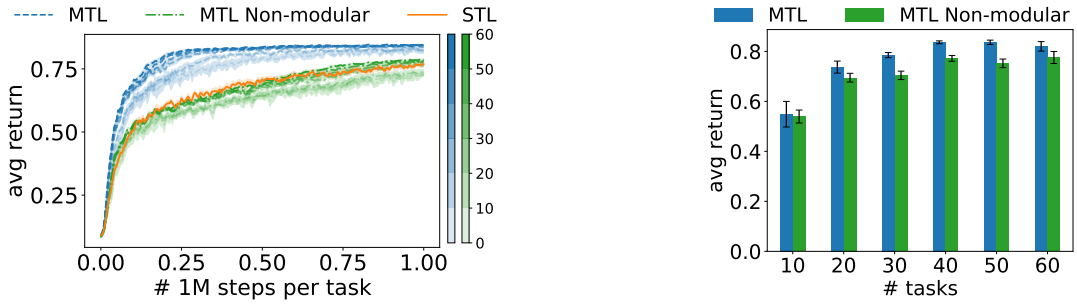


Figure 71: Returns of STL (on 64 Tasks) and MTL (on Various Tasks, Per the Colorbar) on 2-D Tasks

These results suggest that our modular architecture accurately captures the relations across tasks in the form of modules. To verify this, we evaluated the performance of the agent on tasks it had not encountered during training. To construct the network for each task, we again provided the agent with the hard-coded graph structure, but kept the parameters of the modules fixed after MTL training. Figure 71 (right) shows the high zero-shot performance our method achieves, revealing that the modules can be combined in novel ways to solve unseen tasks without any additional training.

Lifelong discovery of modules on discrete 2-D tasks The results of Figure 71 encourage us to envision a lifelong learning agent improving modules over a sequence of tasks and reusing those modules to learn new tasks much more quickly. In this section, we study the ability of Algorithm 12 to achieve this kind of lifelong composition. We consider two instances of our approach: one in which the agent is given the hard-coded graph structures (Comp.+Struct.) and one in which the agent is given no information about how modules are shared and must therefore discover these relations autonomously via discrete search (Comp.+Search). Once the structure is selected for one particular task, data for that task is collected via PPO training, starting from the parameters of the selected modules. Finally, this collected data is used for incorporating knowledge about the current task into the selected modules via batch RL, using data from the current task and all tasks that reuse any of those modules to avoid forgetting. To match our assumptions, unless otherwise stated the initial tasks presented to the agent contain disjoint sets of components.

We compared against the following baselines: *STL*; *P&C* [199], a similar method that keeps shared parameters fixed in a first stage, and pushes new knowledge into the shared parameters in a second stage, but uses a monolithic network, making it much harder to find a solution that works for all tasks; *Online EWC* [199], which continually trains all tasks into a monolithic network with a quadratic penalty for deviating from earlier solutions; and *CLEAR* [154], which uses replay over previous tasks' trajectories with importance sampling and behavior cloning, but trains a standard monolithic network in a single stage. Lifelong baselines were provided with the ground-truth multi-hot indicator of the task components as part of the input.

Figure 72 (left) shows the average learning curves of the lifelong agents trained on all 64 possible 2-D tasks. Even though tasks are trained sequentially, we show the averaged curves to study the ability to accelerate learning: the fact that the curves for our compositional methods are above STL shows that they achieve forward transfer. Note that this acceleration occurs despite our methods using an order of magnitude fewer trainable parameters than STL ($86,350$ vs. $1,080,320$). Additionally, both our methods improve the modules over time, as demonstrated by the trend of increasing zero-shot performance as more tasks are seen (Fig. 72, center). P&C also learns faster than STL, but as we

will see, P&C catastrophically forgets how to solve earlier tasks. Other lifelong baselines perform much worse than STL, since they are designed to keep the solutions to later tasks close to those of earlier tasks, which fails in compositional settings where optimal task policies vary drastically.

In these 2-D tasks, lifelong learners should completely avoid forgetting, since there exist models (compositional and monolithic) that can solve all possible tasks (see Fig. 71). Figure 72 (right) shows the average performance as each task progresses through various stages: the beginning (Zero-shot) and end (Online) of online training, the consolidation of knowledge into the shared parameters (for Comp. and P&C only, Off-line), and the evaluation after all tasks had been trained (Final). Our methods are the only that achieve forward transfer without suffering from any forgetting. Moreover, Comp.+Struct. achieves *backward transfer*: improving the earlier tasks' performance after training on future tasks, as indicated by the increase in performance from the Off-line to Final bars.

To study the flexibility of our method, we evaluated it on a random sequence of tasks, without forcing the initial tasks to be composed of distinct components. Figure 72 shows that this lack of curriculum (Comp.+Search-NC) does not hinder the forward or backward transfer of our approach.

Lifelong discovery of modules on realistic robotic manipulation tasks Having validated that our approach achieves forward transfer without forgetting on the 2-D tasks, we carried out an equivalent evaluation on the more complex and realistic robotic manipulation suite. The architecture was similarly constructed by chaining one module for each type of obstacle, object, and robot arm, and each such module was a multi-layer perceptron. We compare against P&C, the best lifelong baseline in the 2-D tasks, and STL. We empirically found that PPO would output overconfident actions in this continuous-action setting (i.e., high-magnitude outputs unaffected by the stochastic action sampling) when initialized from the existing modules directly, which limited our agent's ability to learn proficient policies. Therefore, we applied minor modifications to the online PPO training mechanism which permitted successful training of all tasks at the cost of often inhibiting zero-shot transfer. Figure 73 (left) shows the learning curves of both our compositional methods. All lifelong agents learned noticeably faster than the base STL agent, and compositional methods were fastest, despite using an order of magnitude fewer trainable parameters than STL ($165,970$ vs. $1,040,304$). Figure 73 (right) also shows that the off-line stage led to a decrease in performance. However, like in the 2-D domain, training on subsequent tasks led to backward transfer, partially

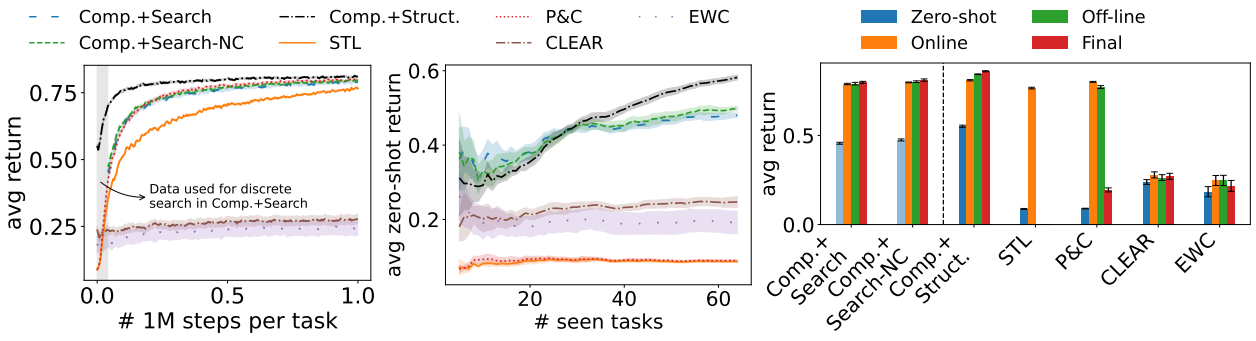


Figure 72: Avg. Returns of STL and Lifelong Agents on 64 Compositional 2-D Discrete Tasks: (Left) Performance During Training on Each Task, (Center) Zero-Shot Performance as More Tasks Are Seen, and (Right) Performance at Various Stages of Training

Figure 73: Avg. Success of STL and Lifelong Agents on 48 Compositional Robot Manipulation Tasks: (Left) During Training on Each Task and (Right) at Various Stages of Training

recovering the performance of the earlier tasks as future tasks were learned. P&C was incapable of retaining knowledge of past tasks, leading to massive catastrophic forgetting.

4.5.3 Unsupervised Hard Example Mining from Videos for Improved Object Detection (DETFLICK).

We evaluate our method on face and pedestrian detection and perform ablation studies analyzing the effect of the hard examples.For pedestrians, we show results on the Caltech dataset [2], while for face detection, we show results on the WIDER Face [3] dataset.

The Caltech Pedestrian Dataset [2] consists of videos taken from a vehicle driving through urban traffic, with about 350k annotated bounding-boxes from 250k video frames.

The WIDER dataset consists of 32,203 images having 393,703 labeled faces in challenging situations of scale, pose and occlusion. The evaluation set of WIDER is divided into *easy*, *medium*, and *hard* sets according to the detection scores of object proposals from EdgeBox [200]. From easy to hard, the faces get smaller and more crowded.

Retraining Detectors with Mined Hard Examples We experimented with two ways to leverage our mined *hard negative* samples. In our initial experiments, a single mini-batch is formed by including one image from the original labeled training dataset and another image sampled from our automatically-mined hard negative video frames. In this way, positive region proposals are sampled from the original training dataset image, based on manual annotation, while negative region proposals are sampled from both the original dataset image and the mined hard negative video frame. Thus, we can *explicitly* force the network to focus on the hard negatives from the mined video frame. However, this method did not produce better results in our initial experiments. An alternate approach was found to be more effective – we simply provided the *pseudo-positives* in the mined video frames as true object annotations during training and *implicitly* allowed the network to pick the hard-negatives. The inclusion of video frames with *hard positives* is more straightforward – we can simply treat them as additional images with object annotations at training time. The models were fine-tuned with and without OHEM, and we consistently chose the setting that gave the best validation results. While OHEM would increase the likelihood of hard negatives being selected in a mini-batch, it would also place extra emphasis on any mislabels in the hard examples. This would magnify the effect of a small amount of label noise and can in some cases decrease the overall performance.

Ablation Settings In addition to the comparisons to the baseline Faster R-CNN detectors, we conduct various ablation studies on the Caltech Pedestrian and WIDER Face datasets to address the effectiveness of hard example mining.

Effect of training iterations. To account for the possible situation where simply training the baseline model longer may result in a gain in performance, we create another baseline by fine-tuning the original model for additional iterations with a lower learning rate, matching the number of training iterations used in our hard example trained models. We refer to this model as "`w/ more iterations`".

Effect of additional video frames. Unlike the baseline detector, our fine-tuned models use additional video frames for training. It's possible that just using the high-confidence detection results on unlabeled video frames as *pseudo-groundtruths* during training is sufficient to boost performance, without correcting the hard negatives using our detector flicker approach. Therefore we train another detector, "`Flickers as Positives`", starting from the baseline model, that takes exactly the same training set as our hard negative model, but where *all* the high-confidence detections on the video frames are used as positive labels.

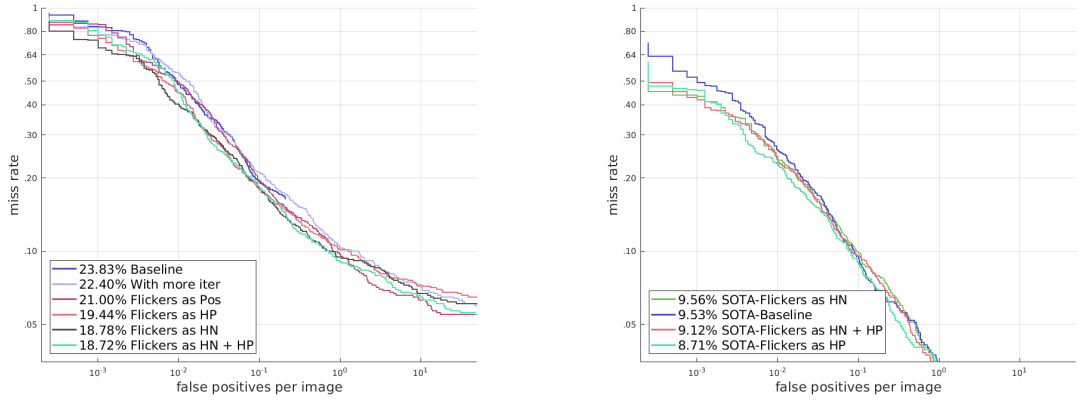
Effect of automatically mined hard examples. We include the results from our proposed method of considering detector flickers as hard negatives and hard positives separately – "`Flickers as HN`" and "`Flickers as HP`". Finally, we report results from fine-tuning the detector on the union of both types of hard examples (`Flickers as HN + HP`).

Pedestrian Detection For our `baseline` model, we train the VGG16-based ***Faster R-CNN*** object detector [88] with OHEM [87] for 150K iterations on the **Caltech Pedestrian** training dataset [2]. We used *all* the frames from set00-set05 (which constitute the training set), irrespective of whether they are flagged as "reasonable" or not by the Caltech meta-data. Following Zhang et al. [201], we set the IoU ratio for RPN training to 0.5, while all the other experimental settings are identical to [88]. The number of labeled Caltech images is 128,419 and our mining provides 14,967 hard negative and 42,914 hard positive frames. We fine-tune the baseline model with hard examples and the annotated examples from the Caltech Pedestrian *training* dataset, with a fixed learning rate of 0.0001 for 60K iterations, using OHEM. We evaluate our model on the Caltech Pedestrian testing dataset under the *reasonable* condition.

The ROC curves of various settings of our models are shown in Figure 74(a) for the faster R-CNN results: using hard negative samples (`Flickers as HN`) and hard positive samples (`Flickers as HP`) improve the performance over the baseline in; using a combination of both gives the best performance. Fine-tuning the existing detector for more iterations gives a modest reduction in log average miss rate, from 23.83% to 22.4%. Using all detections without correcting the hard negatives (`Flickers as Pos`) also gives a small improvement – the extra training data, although noisy, still has some positive contribution during fine-tuning. Our proposed model, fine-tuned with the mined hard negatives (`Flickers as HN`), has a log average miss rate of **18.78%**, which outperforms the `baseline model` by **5.05%**. Fine-tuning with hard positives (`Flickers as HP`) also shows an improvement of **4.39%** over the baseline. Combining both hard positives and hard negatives results in the best performance of **18.72%** log average miss rate.

In Figure 74(b) we report results using the state-of-the-art ***SDS-RCNN*** [202] pedestrian detector [10], showing that `Flickers as HN` improves the original SDS-RCNN results only in the low false

[10] Running the authors' released code from https://github.com/garrickbrazil/SDS-RCNN



(a) (b)

Figure 74: Results on the Caltech Pedestrian Dataset [2] in *Reasonable* Condition: (a) Faster R-CNN Results, (b) State-of-the-Art SDS-RCNN Results

positive regime, while `Flickers as HP` gives the best results. Every 3rd frame is sampled from the Caltech dataset for training the original detector [202], and we keep this setting in our experiments. For SDS-RCNN, there are 42,782 labeled training images while the mining gives us 2,191 hard negative and 177,563 hard positive frames. The inclusion of hard negatives in training (`Flickers as HN`) improves the performance of SDS-RCNN in the low False Positives regime compared to the baseline – the detector learns to eliminate a number of false detections, thereby increasing precision, but it also ends up hurting the recall. Including mined hard positives (`Flickers as HP`) we get the best performance of **8.71%** log average miss rate, outperforming the model using both the mined hard negative and positive samples (`Flickers as HP + HN`), which gets 9.12%.

Face Detection We adopt the Faster R-CNN framework, using VGG16 as the backbone network. We first train a baseline detector starting from an ImageNet pre-trained model, with a fixed learning rate of 0.001 for 80K iterations using the SGD optimizer, where the momentum is 0.9 and weight decay is 0.0005. For hard negatives, the model is fine-tuned for 50k iterations with learning rate 0.0001. For hard positives, and the combination of both types of hard examples, we train longer for 150k iterations. Following the **WIDER Face** protocol, we report Average Precision (AP) values in Table 20 on the three splits – 'Easy', 'Medium' and 'Hard'. OHEM is not used as it was empirically observed to decrease performance.

Fine-tuning the baseline model for more iterations improves performance slightly on the Easy and Medium splits. Naively considering all the high confidence detections as true positives (`Flickers as Positives`) degrades performance substantially across all splits. Hard negative mining, `Flickers as HN`, slightly outperforms the baseline Faster R-CNN detector (w/ more iterations) on the Medium and Hard splits, retaining the same performance of 0.907 AP on the Easy split. Using the mined hard positives, `Flickers as HP`, we observe a significant gain in performance on all three splits. Using both hard positives and hard negatives jointly (`Flickers as HP + HN`) improves over using hard negatives and the baseline, but the improvement is less than the gains from `Flickers as HP`.

Table 20: Average Precision (AP) on the Validation Set of the WIDER Face [3] Benchmark

		Easy	Medium	Hard
Faster R-CNN	Baseline	0.907	0.850	0.492
	w/ more iterations	0.910	0.852	0.493
	Flickers as Positives	0.829	0.790	0.434
	Ours: Flickers as HN	0.909	0.853	0.494
	Ours: Flickers as HP	**0.921**	**0.864**	0.492
	Ours: Flickers as HP + HN	**0.921**	**0.864**	**0.497**

For faces, we additionally experimented with the recent RetinaNet [203] detector as a second high-performance baseline model. Unfortunately, inclusion of the unlabeled data hurt performance slightly using this model, despite the reasonably high purity of the mined examples. While the purity of our mined examples is high, it is not perfect. These incorrect samples would be strongly emphasized by the focal loss used in RetinaNet. Thus, it is possible that while RetinaNet outperforms the Faster R-CNN on standard benchmarks, it may be more susceptible to label noise and thus not a good candidate for our method. In the future, we will investigate different values of the focal loss parameter to see whether this can mitigate the effects of label noise.

Figure 75 shows a qualitative comparison of Faster R-CNN detections for faces (F1-4) and pedestrians (P1-4). The detector fine-tuned with hard negatives (HN) reduces false positives compared to the Baseline (F-1,3,4; P-1,2,3), but can sometimes lower the recall (P4). Hard positives (HP) increases recall (F2, P4) but can also introduce false positives (F4). Using both (HP+HN) the detector is usually able to achieve a good balance.

4.5.4 Automatic adaptation of object detectors to new domains using self-training (STSL).

Datasets Experiments are performed on two challenging scenarios – pedestrian detection from driving videos and face detection from surveillance videos, both of which fit neatly into our paradigm of self-training from large amounts of unlabeled videos and where there exists a significant domain shift between source and target. We select single-category detection tasks like face and pedestrian to avoid the engineering and computational burden of handling multiple categories, and focus on the unsupervised domain adaptation aspect. A summary of the datasets is given in Table 21. Details of the source and target datasets for face and pedestrian detection tasks are summarized here. N.B.– for the unlabeled target train sets, the #images and #annotations are unknown.

Face: WIDER $\rightarrow$ CS6. The WIDER dataset [3] is the source domain, consisting of labeled faces in still images downloaded from the internet with a wide variety of scale, pose and occlusion. The baseline detector is trained on the WIDER Train split, which has 12,880 images and 159,424 annotated faces. The target domain consists of 179 surveillance videos from CS6, which is a subset of the IJB-S benchmark [1]. CS6 provides a considerable shift from WIDER, with faces being mostly low-resolution and often occluded, and the imagery being of low picture quality, suffering from camera shake and motion blurs. The video clips are on average of 5 minutes at 30 fps, with some exceptionally long clips running for over an hour. We selected 86 videos to form the unlabeled target train set (*CS6-Train*). A test set of 80 labeled videos, containing about 70,022 images and 217,827 face annotations, is used to evaluate the performance of the methods (*CS6-Test*).

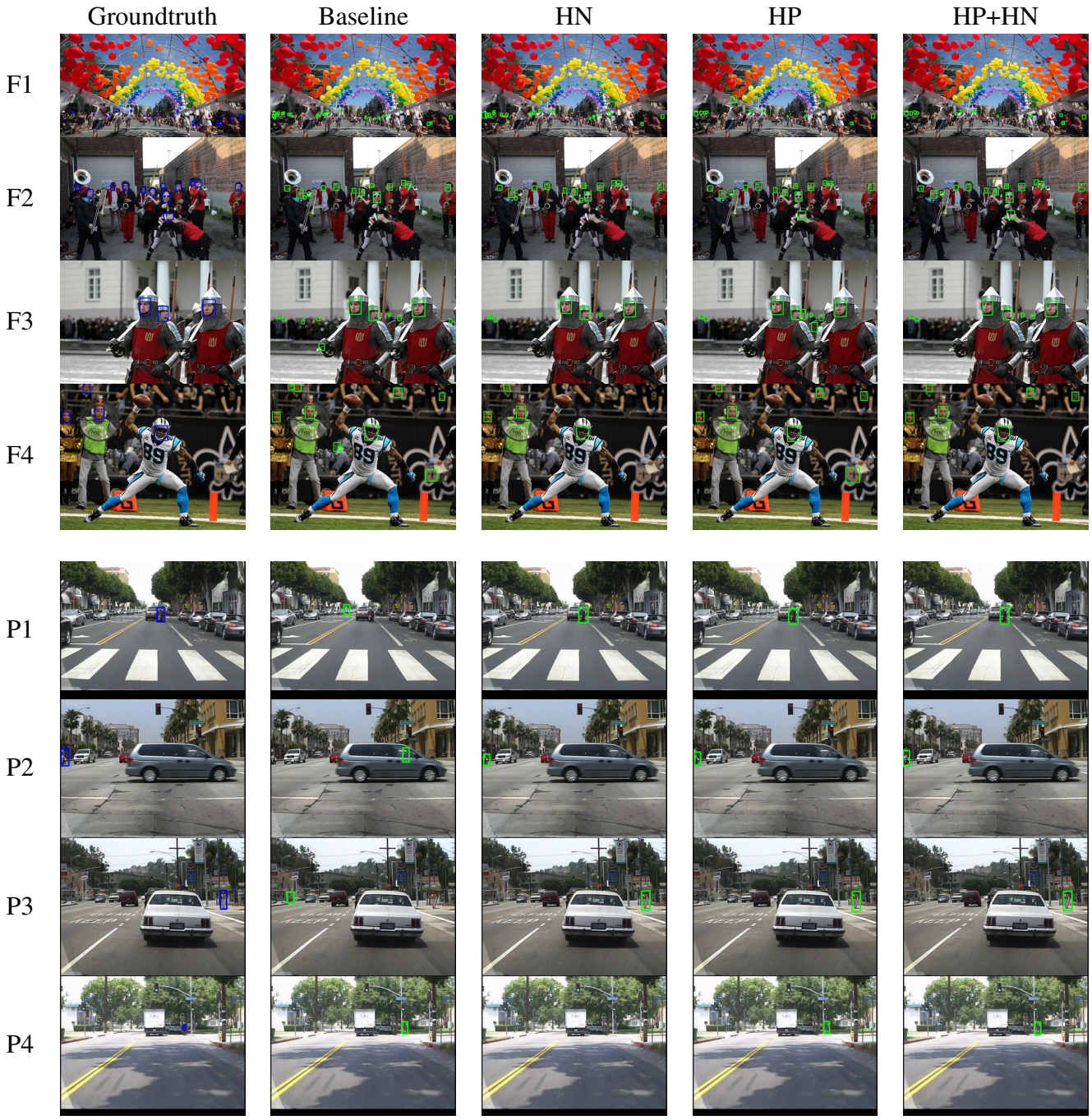


Figure 75: Qualitative Comparison of Faster R-CNN Detections for Faces (F1-4) and Pedestrians (P1-4).

Pedestrian: BDD(*clear,daytime*) → BDD(*rest*). The Berkeley Deep Drive 100k (BDD-100k) dataset [204] consists of 100,000 driving videos from a wide variety of scenes, weather conditions and time of day, creating a challenging and realistic scenario for domain adaptation. Each video clip is of 40 seconds duration at 30 fps; one frame out of every video is manually annotated. The source domain consists of clear, daytime conditions (*BDD(clear,daytime)*) and the target domain consists

Table 21: STSL - Dataset Summary

Dataset	# images	# annots	# videos
WIDER	12,880	159,424	-
CS6-Train	-	-	86
CS6-Test	70,022	217,827	80
BDD-Source	12,477	16,784	12,477
BDD-Target-Train	-	-	18,000
BDD-Target-Test	8,236	10,814	8,236

Table 22: STSL - Pseudo-labels Summary

Method	# images	# annots
CS6-Det	38,514	109,314
CS6-HP	15,092	84,662
CS6-Track	15,092	32,711
BDD-Det	100,001	205,336
BDD-Track	100,001	222,755
BDD-HP	100,001	362,936

of all other conditions including night-time, rainy, cloudy, etc (*BDD(rest)*). There are 12,477 labeled images forming *BDD-Source-Train*, containing 217k pedestrian annotations. We use 18k videos as the unlabeled *BDD-Target-Train* set, having approximately 21.6 million video frames (not all of which would contain pedestrians, naturally). The *BDD-Target-Test* set is comprised of 8,236 labeled images with 16,784 pedestrian annotations from *BDD(rest)*.

Baselines and Ablations We consider the following methods as our baselines:

Baseline source. Detector trained on only the labeled source data – WIDER for faces and BDD(*clear,daytime*) for pedestrians.

Pseudo-labels from detections. High-confidence detections on the target training set are considered as training labels, followed by joint re-training of the baseline source detector. This is the naive baseline for acquiring pseudo-labels, denoted as `Det` in the results tables.

Pseudo-labels from tracking. Incorporating temporal consistency using a tracker and adding them into the set of pseudo-labels was referred to as *"Hard Positives"* by Jin et al. [11]; we adopt their nomenclature and refer to this as `HP`. As an ablation, we exclude detector results and keep just the *tracker-only* pseudo-labels for training (`Track`). Table 22 summarizes the details of the automatically gathered pseudo-labels, listing the number of images and object annotations obtained on the unlabeled CS6-Train and BDD-Target-Train videos. All the pseudo-labels obtained from the CS6 videos are used in re-training. For BDD, due to the massive number of videos, 100K frames were sub-sampled to form the training set. Note that using temporal constraints (`HP`) removes spurious isolated detections in addition to adding missed objects, resulting in an overall decrease in data when compared to *Det* for CS6.

Soft labels for distillation. The *label interpolation* method as detailed in Sec. 3.4.3 is denoted as `Label-smooth`, and we show the effect of varying λ on the validation set. Cross-domain score

distribution mapping is referred to as `score-remap` and constrained hard examples as `HP-`*cons* in the results tables.

Domain adversarial Faster-RCNN. While there are several domain adversarial methods such as ADDA [205] and CyCADA [206] for object *recognition*, we select Chen et al. [207] as the only method, to our knowledge, that has been integrated into the Faster R-CNN *detector*. Chen et al. [207] formulate the adversarial domain discriminator [208] with three separate losses – (i) predicting the domain label from the convolutional features (pre-ROI-pooling) of the entire image; (ii) predicting the domain label from the feature-representation of each proposed ROI; (iii) a consistency term between the image-level and ROI-level predictions. The region-proposals for the ROI-level loss are obtained from the Region Proposal Network (RPN) branch of the Faster R-CNN. In our experiments, we denote these models as – `DA-`*im* which applies the domain discriminator at the image level and `DA-`*im-roi*, which additionally has the instance-level discriminator and consistency term.

Training and Evaluation We use the standard Faster R-CNN detector [88] for all our experiments, from a PyTorch implementation of the Detectron framework [209]. An ImageNet-pre-trained ResNet-50 network is used as a backbone, with ROI-Align region pooling. For faces, the baseline is trained for 80k iterations, starting from a learning rate of 0.001, dropping to 0.0001 at 50k, using 4 GPUs and a batch-size of 512. For pedestrians, the baseline was trained for 70k iterations, starting with a learning rate of 0.001 and dropping to 0.0001 at 50k. During training, face images were resized to be 800 pixels and pedestrian images were resized to be 500 pixels on the smaller side. Re-training for the target domain is always done jointly, using a single GPU – a training mini-batch is formed with samples from a labeled source image and a pseudo-labeled target image. In practice, we sample images alternately from source and target, fix 64 regions to be sampled from each image, and accumulate gradients over the two images before updating the model parameters. Domain adversarial models were implemented following Chen et al. [207], keeping their default hyper-parameter values.

Since unsupervised learning considers no labels *at all* on the target domain, we cannot set hyper-parameters or do best model selection based on a labeled validation set. The re-training for *all* the face models were stopped at the 10k iteration, while *all* the pedestrian models were stopped at the 30k iteration. For evaluating performance, to account for stochasticity in the training procedure, we do 5 rounds of training and evaluate each model on the labeled images from the test set. We use the MS-COCO toolkit as a consistent evaluation metric for both face and pedestrian detection, reporting Average Precision (AP) at an IoU threshold of 0.5.

Figure 76 showcases qualitative results of STSL. The domain adapted methods pick up prominent objects missed by the baseline (*cols 1,3-5*). On pedestrians (*cols 3-5*) the detection scores from DA is usually lower than our models', leading to lower overall performance despite correct localization.

Face detection results The results on adapting from labeled WIDER Faces still-images to unlabeled CS6 surveillance video imagery are shown in Table 23.

Effect of pseudo-labels. The *baseline* detector, trained on WIDER Face, gets an AP of 15.66 on CS6-Test, which underscores the domain shift between WIDER and the surveillance video domain. Using only the high-confidence detections (θ=0.5) as training samples, *CS6-*`Det`, boosts performance to 17.29 AP. Using only samples from the tracker and ignoring all pseudo-labels from the detector, *CS6-*`Track`, brings down the performance to 11.73 AP. This can be partly attributed to the fact that we may miss a lot of actual faces in an image if we choose to train only on faces

Figure 76: STSL - Qualitative results (best zoomed-in). (a) Baseline; (b) HP [11]; (c) *Ours*; (d) DA [207].

picked up by tracking alone. The combination of both tracking and detection results for training, *CS6*-HP, gives slightly better performance of 17.31 AP. This is a significant boost over the model trained on WIDER-Face: 15.66 $\rightarrow$ 17.31.

Effect of soft-labels. Incorporating soft target labels gives a consistent gain over the default hard labels, as seen in the Label-smooth numbers in Table 23. The effect of varying the distillation weight λ results in some fluctuation in performance – $AP_{\lambda=0.3}$ is 19.89, $AP_{\lambda=0.5}$ is 19.56 and $AP_{\lambda=0.7}$ is 20.80. Using the completely parameter-free methods we get 19.12 from score histogram remapping (score-remap) and a slightly higher number, 20.65, from HP-*cons*. Both are comparable to distillation with $\lambda = 0.7$.

Comparison to domain discriminator. The domain adversarial method (DA) gives a high performance on CS6 Test with an AP of 21.02 at the image-level (DA-*im*) and 22.18 with the instance-level adaptation included (DA-*im-roi*). Our best numbers (20.80, 20.65) are comparable to this, given the variance over 5 rounds of training.

Pedestrian detection results The results on adapting from BDD-Source images from clear, daytime videos to unconstrained settings in BDD-Target are shown in Table 24. In addition to a new task, the target domain of BDD-Pedestrians provides a more challenging situation than CS6. The target domain now consists of multiple modes of appearance – snowy, rainy, cloudy, night-time, dusk, etc.; and various combinations thereof.

Effect of pseudo-labels. The *baseline* model gets a fairly low AP of 15.21, which is reasonable given the large domain shift from source to target. *BDD*-Det, which involves training with only the high-confidence detections (threshold $\theta = 0.8$), improves significantly over the baseline with an AP of 26.16. Using only the tracker results as pseudo-labels, *BDD*-Track, gives similar performance (26.28). *BDD*-HP, which combines pseudo-labels from both detection and tracking, gives the best performance among these (27.11). This is a significant boost over the baseline: 15.21 $\rightarrow$ 27.11.

Table 23: WIDER → CS6: Average Precision (AP) on CS6 Surveillance Videos

Method	**AP** (mean ± std)
Baseline: WIDER	15.66 ± 0.00
CS6-Det	17.29 ± 0.85
CS6-Track	11.73 ± 0.77
CS6-HP [11]	17.31 ± 0.60
CS6-Label-smooth($\lambda = 0.3$)	19.89 ± 0.92
CS6-Label-smooth($\lambda = 0.5$)	19.56 ± 1.53
CS6-Label-smooth($\lambda = 0.7$)	**20.80** ± **1.34**
Ours: CS6-score-remap	19.12 ± 1.29
Ours: CS6-HP-*cons*	**20.65** ± **1.62**
CS6-DA-*im* [207]	21.02 ± 0.96
CS6-DA-*im-roi* [207]	**22.18** ± **1.20**

Effect of soft-labels. Using soft labels via `Label-smooth` improves results further (27.11 → 28.59), with performance fluctuating slightly with different values of the λ hyper-parameter – $AP_{\lambda=0.3}$ is 28.59, $AP_{\lambda=0.5}$ is 28.38 and $AP_{\lambda=0.7}$ is 28.47. Creating soft-labels via score histogram matching (`score-remap`), we get an AP of 28.02. Emphasizing tracker-only samples while constraining identical behaviour on detector training samples (`HP`-*cons*) gives 28.43. Again, both these methods are comparable in performance to using `Label-smooth`, with the advantage of not having to set the λ hyper-parameter.

Comparison to domain discriminator. Adapting to the BDD-Target domain was challenging for the domain adversarial (DA) models [207], most likely due to the multiple complex appearance changes, unlike the WIDER→CS6 shift which has a more homogeneous target domain. The image-level adaptation (`DA`-*im*) models gave 23.65 AP – a significant improvement over the baseline AP of 15.21. We had difficulties getting the `DA`-*im-roi* model to converge during training. Using the pseudo-labels from BDD-HP for class-balanced sampling of the ROIs during training had a stabilizing effect (denoted by BDD-`DA`-*im-roi**). This gives 23.69 AP. Overall our results from training with soft pseudo-labels are better than [207] on this dataset by ∼5 in terms of AP.

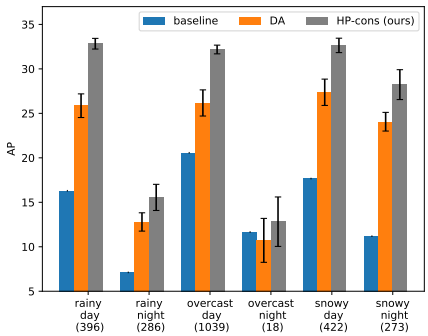


Figure 77: BDD(*rest*) Sub-Domains: Performance of the *baseline* Model, Domain Adversarial Model (*DA*) and Our Method (*HP-cons*). The number of images in each sub-domain is written in parentheses below.

Table 24: STSL - BDD(*clear,daytime*) → BDD(*rest*)

Method	**AP** (mean ± std)
Baseline: BDD(*clear,daytime*)	15.21 ± 0.00
BDD-Det	26.16 ± 0.24
BDD-Track	26.28 ± 0.35
BDD-HP [11]	27.11 ± 0.54
BDD-Label-smooth($\lambda = 0.3$)	**28.59 ± 0.67**
BDD-Label-smooth($\lambda = 0.5$)	28.38 ± 0.62
BDD-Label-smooth($\lambda = 0.7$)	28.47 ± 0.41
Ours: BDD-score-remap	*28.02 ± 0.32*
Ours: BDD-HP-*cons*	**28.43 ± 0.51**
BDD-DA-*im* [207]	23.65 ± 0.57
BDD-DA-*im-roi**	**23.69 ± 0.93**

Table 25: STSL - Sensitivity to Detector Confidence Threshold for Target-Domain Pseudo-labels, Evaluated for the *HP* Model

$\theta \rightarrow$	0.5	0.6	0.7	0.8	0.9	$\theta_{\mathcal{S}\rightarrow\mathcal{T}}$
CS6-Test	17.31	15.91	14.93	15.63	11.69	*16.71*
BDD-Test	27.23	27.68	27.30	27.11	25.85	*27.11*

Results on sub-domains. The BDD-Target domain implicitly contains a large number of *sub-domains* such as rainy, foggy, night-time, dusk, etc.. We compare the performance of three representative models – baseline, domain adversarial (`DA`-*im*) and our soft-labeling method (we pick `HP`-*cons* as representative) on a set of such implicit sub-domains in BDD-Target-Test for a fine-grained performance analysis (Fig. 77). Night-time images clearly degrade performance for all the models. Overall both domain adaptive methods improve significantly over the baseline, with `HP`-*cons* consistently outperforming `DA`. It is possible that higher performance from DA can be obtained by some dataset-specific tuning of hyper-parameters on a validation set of *labeled* target-domain data.

Automatic threshold selection The hyper-parameter θ that thresholds the high-confidence detections can be set without manual inspection of the target domain. We can pick a threshold $\theta_{\mathcal{S}}$ on *labeled source* data for a desired level of precision, say 0.95. Using score histogram mapping $\mathcal{S} \rightarrow \mathcal{T}$ (Sec 3.4.3, Fig. 17), we can map $\theta_{\mathcal{S}}$ to the *unlabeled target* domain as $\theta_{\mathcal{T}}$. These results are shown in Table 25. The thresholds selected based on visual inspection of 5 videos are 0.5 for faces (17.31 AP) and 0.8 for pedestrians (27.11 AP). The performance from automatically set $\theta_{\mathcal{S}\rightarrow\mathcal{T}}$ is very close – AP of 16.71 on CS6 and 27.11 on BDD.

4.5.5 Half&Half: New Tasks and Benchmarks for Studying Visual Common Sense (HNH).

Image-to-Label Our task for this benchmark is to identify object categories that are likely to be present in the right half by observing only the left half. We formulate this as a multi-category

classification problem.

We define two types of classifiers: **symmetric** and **anti-symmetric**. A symmetric classifier is a standard CNN trained on the left-half image to predict the labels of objects in the *left half* itself, which is equivalent to a traditional classifier. Meanwhile, an anti-symmetric classifier is trained on the same set of left-half images, but with object categories present in the right half as the target labels.

For training the classifiers, we use the training split provided by the benchmark. Given the set of all left-half images, we train a CNN (ResNet-50 [210] pretrained on ImageNet) to predict the presence of object categories. The last FC layer is modified to match the 79 object categories we use according to the classifier (symmetric or anti-symmetric). If there are multiple categories for an image, we duplicate the left-half image in the training set and assign an individual category to each of the left-half images. We follow this approach so as to maintain consistent behavior with the benchmark-setting, where our candidate list only contains a single correct object category.

From the trained network, we obtain the posterior probability distribution over all 79 categories in the MS-COCO dataset. We evaluate the performance of our context driven model on the benchmark by computing the ranking of the five candidate categories in the candidate list according to their posterior probabilities.

Tab. 26 compares symmetric and anti-symmetric classifiers, as well as a MLP baseline using GIST [211].

Table 26: HNH - Evaluations on the Image-to-Label Benchmark

Classifier	Rank-1 Acc.	MRR
Random Chance	20.0 %	0.457
MLP (GIST)	42.0%	0.635
Symmetric	58.7%	0.707
Anti-Symmetric	74.3%	0.855

Label-to-Image For the Label-to-Image benchmark, our goal is to rank the candidate images based on the likelihood of containing the given query object. We propose following two methods.

Indirect training In this case, we use any classifier trained on the Image-to-Labeltask to compute posterior probabilities for the query object. Based on these posteriors, candidates are ranked. We directly use the anti-symmetric classifier trained for the Image-to-Labelbenchmark.

Direct training In the second method, we train a CNN to *directly* compare the given candidate images conditioned on the query label. We formalize this as a classification problem where, given the candidate set, the objective is to predict the most likely image to contain the query label. For each image in the candidate set, we compute a *class score* for the query label. To do this, we consider the output of the classification layer of a CNN. We then normalize the class scores across candidate images. This reflects the probability distribution among the candidate images given the query label. Finally, ten candidate images are ranked according to their respective posterior

probabilities for the query label. We use the same ResNet-50 model (ImageNet pretrained) as our base classifier.

The direct and indirect classifiers are compared in Tab. 27. Direct training provides a noticeable gain, which suggests it is beneficial to have a training objective more closely aligned with the actual evaluation protocol.

Table 27: HNH - Evaluations on the Label-to-Image Benchmark

Classifier	Rank-1 Acc.	MRR
Indirect	44.7%	0.624
Direct	46.6%	0.646

Image-to-Image For this task, given a query image and the 10 gallery images, the goal is to find the correct match coming from the same image. As two **baselines**, we compute the L2 distance of feature vectors from the last fully connected layer of a ResNet-18 [210] between the query image and each of the gallery images. We use models pre-trained on ImageNet [212] and Places365 [213].

In addition to the baselines, we also train networks with two different metric learning techniques. Suppose we have feature vectors, $\mathbf{x} \in \mathcal{R}^{D\times 1}$, $\mathbf{y} \in \mathcal{R}^{D\times 1}$, computed from the backbone network. With **bilinear metric learning**, a similarity score between $\mathbf{x}$ and $\mathbf{y}$ is computed by $\mathbf{x}^\intercal \mathbf{W} \mathbf{y}$ where $\mathbf{W} \in \mathcal{R}^{D\text{x}D}$ can be trained. We also train networks with **symmetric metric learning** that learns $\mathbf{L} \in \mathcal{R}^{D\text{x}D}$ in $(\mathbf{L}\mathbf{x})^\intercal (\mathbf{L}\mathbf{y})$.

All of our networks are trained using triplet loss [214], where a triplet is generated with a query image, the corresponding correct match, and one of the 9 negatives. Each model is then trained such that a query image is closer to the correct match than to the negative. We first train networks by freezing the parameters in the backbone network. After $\mathbf{W}$ and $\mathbf{L}$ are learned, we also fine-tune the entire network.

Table 28: HNH - Evaluations on the Image-to-Image Benchmark

Pre-trained	f.t.?	L2	Symm. Metric	Bilinear Metric
ImageNet		47.3%	54.0%	54.1%
ImageNet	✓	65.3%	64.8%	70.0%
Places365		56.2%	63.1%	65.1%
Places365	✓	67.2%	67.7%	69.0%

Results and comparisons are summarized in Tab. 28. A few observations can be made: (1) Place365 offers better pre-training compared to ImageNet for our problems; (2) various metric learning technique all help significantly compared to direct L2 distances; and (3) fine-tuning the backbone network offers consisting improvements.

(a) **(b)** **(c)**

Figure 78: HNH - Example of a navigation task built from the Active Vision Dataset. Query: "toilet"; Correct answer: (c).

Preliminary Results for Visual Navigation In this section, we demonstrate how models trained on our benchmarks can be useful for visual navigation applications. For this preliminary evaluation, we build a small test set using the Active Vision dataset [215], originally designed for indoor navigation. We formulate the problem statement as: "Which is the best scene to find the target object?". We follow the Label-to-Imagetask formulation to create an approximation of visual navigation task by sampling a target object label and three candidate images (one correct, two wrong) to create a problem (see an example in Fig. 78).

Model trained on our Label-to-Imagebenchmark (the direct training variant) achieves 68% accuracy and human performance is at 98.3%. We conduct a human study with 6 participants. While our model does show significant advantage over chance performance (33%), there is still a large gap towards human performance, highlighting the importance of future research in this direction.

4.5.6 Efficient Lifelong Inverse Reinforcement Learning.

Evaluation domains We evaluated ELIRL on two environments, chosen to allow us to create arbitrarily many tasks with distinct reward functions. This also gives us known rewards as ground truth. No previous multi-task IRL method was tested on such a large task set, nor on tasks with varying state spaces as we do.

- **Objectworld** Similar to the environment presented by Levine et al. [216], Objectworld is a 32×32 grid populated by colored objects in random cells. Each object has one of five outer colors and one of two inner colors, and induces a constant reward on its surrounding 5×5 grid. We generated 100 tasks by randomly choosing 2–4 outer colors, and assigning to each a reward sampled uniformly from $[-10, 5]$; the inner colors are distractor features. The agent's goal is then to move toward objects with "good" (positive) colors and away from objects with "bad" (negative) colors. Ideally, each column of $\boldsymbol{L}$ would learn the impact field around one color, and the $\boldsymbol{s}^{(t)}$'s would encode how good or bad each color is in each task. There are $d = 31(5 + 2)$ features, representing the distance to the nearest object with each outer and inner color, discretized as binary indicators of whether the distance is less than 1–31. The agent can choose to move along the four cardinal directions or stay in place.

  Highway Highway simulations have been used to test various IRL methods [217, 216]. We simulate the behavior of 100 different drivers on a three-lane highway in which they can drive at four speeds. Each driver prefers either the left or the right lane, and either the second or fourth speed. Each driver's weight for those two factors is sampled uniformly from $[0, 5]$. Intuitively, each column of $\boldsymbol{L}$ should learn a speed or lane, and the $\boldsymbol{s}^{(t)}$'s should encode the

drivers' preferences over them. There are $d = 4 + 3 + 64$ features, representing the current speed and lane, and the distances to the nearest cars in each lane in front and back, discretized in the same manner as Objectworld. Each time step, drivers can choose to move left or right, speed up or slow down, or maintain their current speed and lane.

In both environments, the agent's chosen action has a 70% probability of success and a 30% probability of a random outcome. The reward is discounted with each time step by a factor of $\gamma = 0.9$.

Evaluation procedure For each task, we created an instance of the MDP by placing the objects in random locations. We solved the MDP for the true optimal policy, and generated simulated user trajectories following this policy. Then, we gave the IRL algorithms the MDP$\backslash \boldsymbol{r}$ and the trajectories to estimate the reward $\boldsymbol{r}$. We compared the learned reward function with the true reward function by standardizing both and computing the ℓ_2-norm of their difference. Then, we trained a policy using the learned reward function, and compared its expected return to that obtained by a policy trained using the true reward.

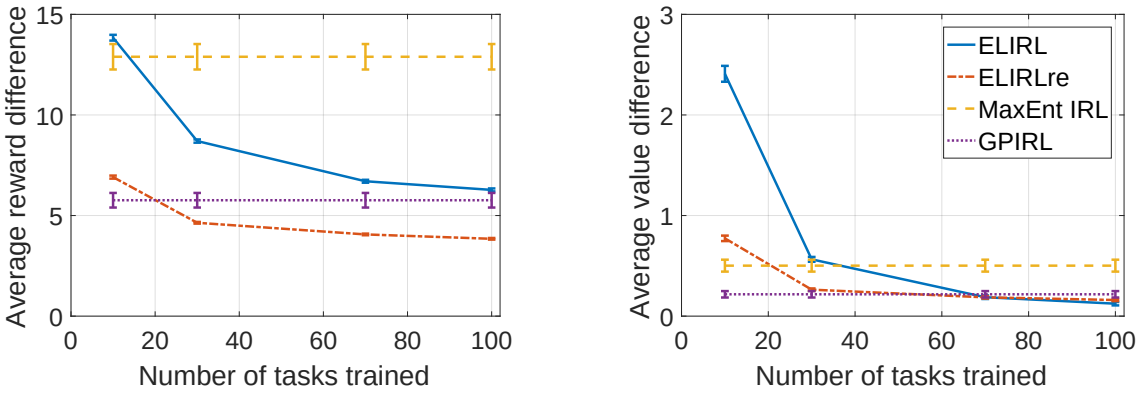
We tested ELIRL using $\boldsymbol{L}$ trained on various subsets of tasks, ranging from 10 to 100 tasks. At each testing step, we evaluated performance of *all* 100 tasks; this includes as a subset evaluating all previously observed tasks, but it is significantly more difficult because the latent basis $\boldsymbol{L}$, which is trained only on the initial tasks, must generalize to future tasks. The single-task learners were trained on all tasks, and we measured their average performance across all tasks. All learners were given $n_t = 32$ trajectories for Objectworld and $n_t = 256$ trajectories for Highway, all of length $H = 16$. We chose the size k of $\boldsymbol{L}$ via domain knowledge, and initialized $\boldsymbol{L}$ sequentially with the $\boldsymbol{\alpha}^{(t)}$'s of the first k tasks. We measured performance on a new random instance of the MDP for each task, so as not to conflate overfitting the training environment with high performance. Results were averaged over 20 trials, each using a random task ordering.

We compared ELIRL with both the original (ELIRL) and re-optimized (ELIRLre) $\boldsymbol{s}^{(t)}$ vectors to MaxEnt IRL (the base learner) and GPIRL [216] (a strong single-task baseline). None of the existing multi-task IRL methods were suitable for this experimental setting—other methods assume a shared state space and are prohibitively expensive for more than a few tasks [218, 219, 220], or only learn different experts' approaches to a single task [221].

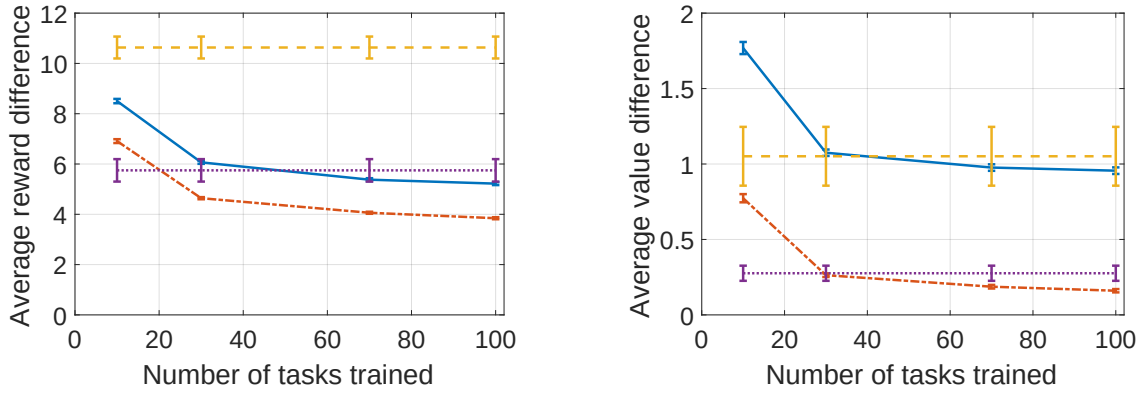
Results Figure 79 shows the advantage of sharing knowledge among IRL tasks. ELIRL learned the reward functions more accurately than its base learner, MaxEnt IRL, after sufficient tasks were used to train the knowledge base $\boldsymbol{L}$. This directly translated to increased performance of the policy trained using the learned reward function. Moreover, the $\boldsymbol{s}^{(t)}$ re-optimization allowed ELIRLre to outperform GPIRL, by making use of the most updated knowledge.

As shown in Table 29, ELIRL requires little extra training time versus MaxEnt IRL, even with the optional $\boldsymbol{s}^{(t)}$ re-optimization, and runs significantly faster than GPIRL. The re-optimization's additional time is nearly imperceptible. This signifies a clear advantage for ELIRL when learning multiple tasks in real-time.

In order to analyze how ELIRL captures the latent structure underlying the tasks, we created new instances of Objectworld and used a single learned latent component as the reward of each new MDP (i.e., a column of $\boldsymbol{L}$, which can be treated as a latent reward function factor). Figure 80



(a) Objectworld



(b) Highway

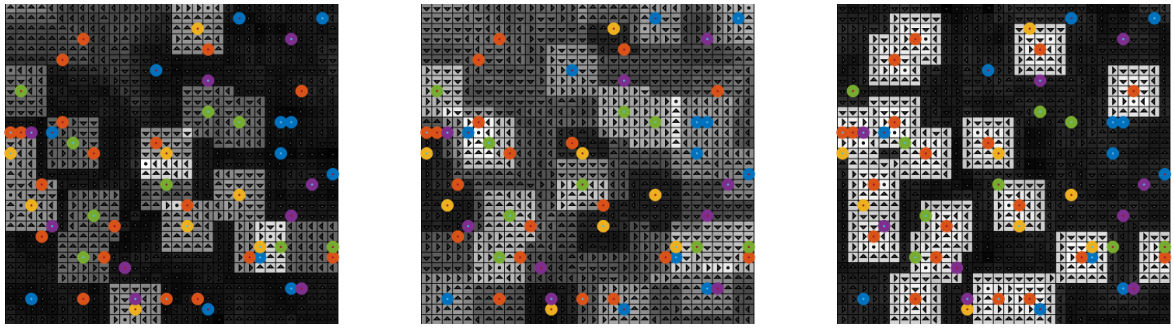
Figure 79: Average Reward and Value Difference in the Lifelong Setting

Table 29: Average Learning Time Per Task

	Objectworld (sec)	Highway (sec)
ELIRL	17.055 ± 0.091	21.438 ± 0.173
ELIRLre	17.068 ± 0.091	21.440 ± 0.173
MaxEnt IRL	16.572 ± 0.407	18.283 ± 0.775
GPIRL	1008.181 ± 67.261	392.117 ± 18.484

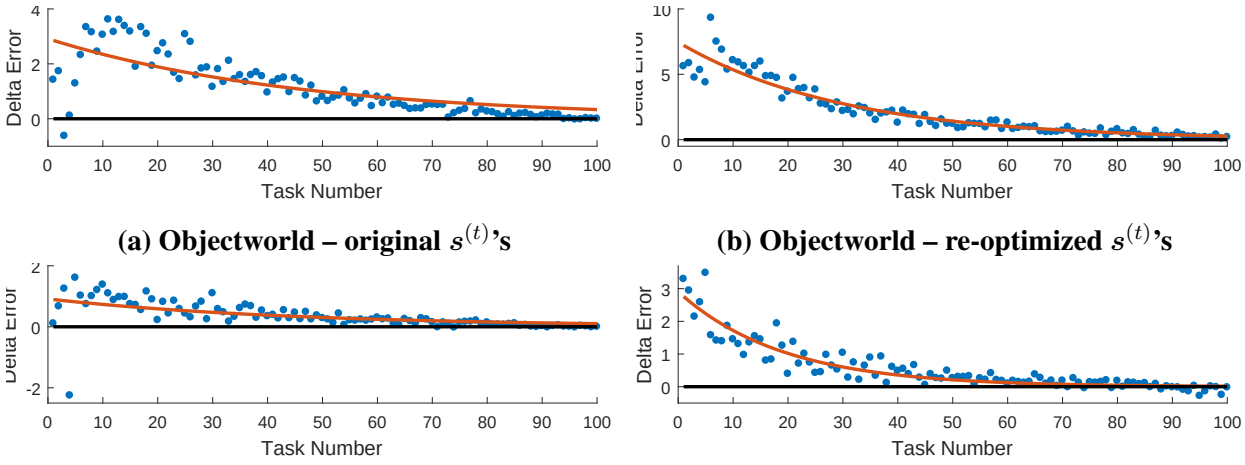
shows example latent components learned by the algorithm, revealing that each latent component represents the 5×5 grid around a particular color or small subset of the colors.

We also examined how performance on the earliest tasks changed during the lifelong learning process. Recall that as ELIRL learns new tasks, the shared knowledge in $\boldsymbol{L}$ continually changes. Consequently, the modeled reward functions for all tasks continue to be refined automatically over time, without retraining on the tasks. To measure this effect of "reverse transfer" [35], we compared the performance on each task when it was first encountered to its performance after learning all tasks, averaged over 20 random task orders. Figure 81 reveals that ELIRL improves previous tasks' performance as $\boldsymbol{L}$ is refined, achieving reverse transfer in IRL. Reverse transfer was further improved by the $\boldsymbol{s}^{(t)}$ re-optimization.



(a) Green and yellow (b) Green, blue, yellow (c) Orange

Figure 80: Example Latent Reward Functions From Objectworld Learned by ELIRL



(a) Objectworld – original $s^{(t)}$'s (b) Objectworld – re-optimized $s^{(t)}$'s

(c) Highway – original $s^{(t)}$'s (d) Highway – re-optimized $s^{(t)}$'s

Figure 81: Reverse Transfer Between When a Task Was First Trained and After the Full Model Had Been Trained

4.5.7 Task-agnostic lifelong learning using high-level shared sets of features (SHELS).

This section evaluates the effectiveness of our framework separately for both novelty detection and novelty accommodation. We then demonstrate how our SHELS representation and approach enable an iterative novelty detection and accommodation process in a class-incremental continual learning scenario without labeled class boundaries.[11]

Datasets We evaluate our framework using MNIST [222], FashionMNIST (FMNIST; [223]), CIFAR10 [224], SVHN [225], and GTSRB [226]. MNIST, FMNIST, CIFAR10, and SVHN each consist of 10 classes, while GTSRB consists of 43 classes.

Networks and training details All experiments use DNN architectures that consist of ReLU convolutional layers followed by ReLU fully-connected layers and cosine normalization. For

[11] Source code supporting re-creation of all experiments can be found at https://github.com/Lifelong-ML/SHELS.

MNIST, FMNIST, CIFAR10 (within-dataset), SVHN, and GTSRB, the architecture consists of six convolutional layers followed by one or more linear layers and a final cosine normalization layer. Across-datasets experiments with CIFAR10 as the ID classes use VGG16 pretrained on ImageNet, replacing the final fully-connected layer with a cosine normalization layer. We use the Adam optimizer for all datasets except for CIFAR10 across datasets, for which we use SGD as it achieves higher ID performance. Additionally, in all experiments, we compute thresholds and tune hyperparameters using only ID data.

Novelty detection We consider two experimental setups of varying difficulties to evaluate our method. The standard evaluation for OOD detection in prior work considers novelty detection *across datasets*, where ID data differs significantly from OOD data. In this case, one entire dataset is considered ID, and an entirely different dataset is selected as OOD. Additionally, we propose evaluating novelty detection *within datasets*, where ID and OOD data share similar characteristics, requiring novelty detection approaches to learn more fine-grained differences than in the across-datasets experiments. For this case, we randomly sample C classes from a single dataset as ID, and use the remaining classes from the dataset as OOD. For MNIST, FMNIST, CIFAR10, and SVHN, we sample $C = 5$ ID classes and use the remaining 5 as novel. For GTSRB, we sample $C = 23$ and use the remaining 20 classes as OOD data.

In all experiments, we train a classification model over an ID train and validation set. We then evaluate the model on both ID test data and OOD test data. Using this procedure, we compare our OOD detection approach to a baseline of the best performing state-of-the-art OOD detector [227], which outperforms alternative methods including ODIN and Mahalanobis. We evaluate performance using the following measures:

- OOD detection accuracy—the ability to correctly classify an OOD data point as novel
- ID classification accuracy—the ability to correctly determine the class of an ID data point
- Combined ID and OOD accuracy—the ability to classify a data point as the correct ID class or as OOD
- AUROC—the ability to discriminate between ID and OOD data over the combined ID and OOD datasets.

To aid generalization of results, all experiments are repeated over 10 trials with random seeds, within-dataset experiments use different sets of ID classes for each seed, and we perform statistical significance analyses over the trials.

Detection and classification accuracies for across-datasets experiments are shown in Figure 82, with corresponding AUROC measures given in Table 30. The results show a trade-off between the learnt models' abilities to perform OOD detection and maintain ID classification accuracy. To analyze this trade-off, we performed two-tailed unpaired Student's t-tests between the baseline and our SHELS approach, using a Holm-Bonferroni adjustment to account for testing multiple measures (all test statistics, p-values, and corrections are reported in the Appendix of the original paper). The t-tests show that SHELS significantly improves OOD performance with respect to the baseline in half of the cases—MNIST-FMNIST, SVHN-CIFAR10, SVHN-GTSRB, CIFAR10-SVHN—with comparable performance otherwise. This improvement in OOD detection is further supported by the AUROCs (Table 30), which shows that SHELS outperforms the baseline over a majority of experiments. In contrast, t-tests show the baseline outperforms our method in ID accuracy in all cases except when CIFAR10 is used as the ID data, where performance is comparable. Note that the

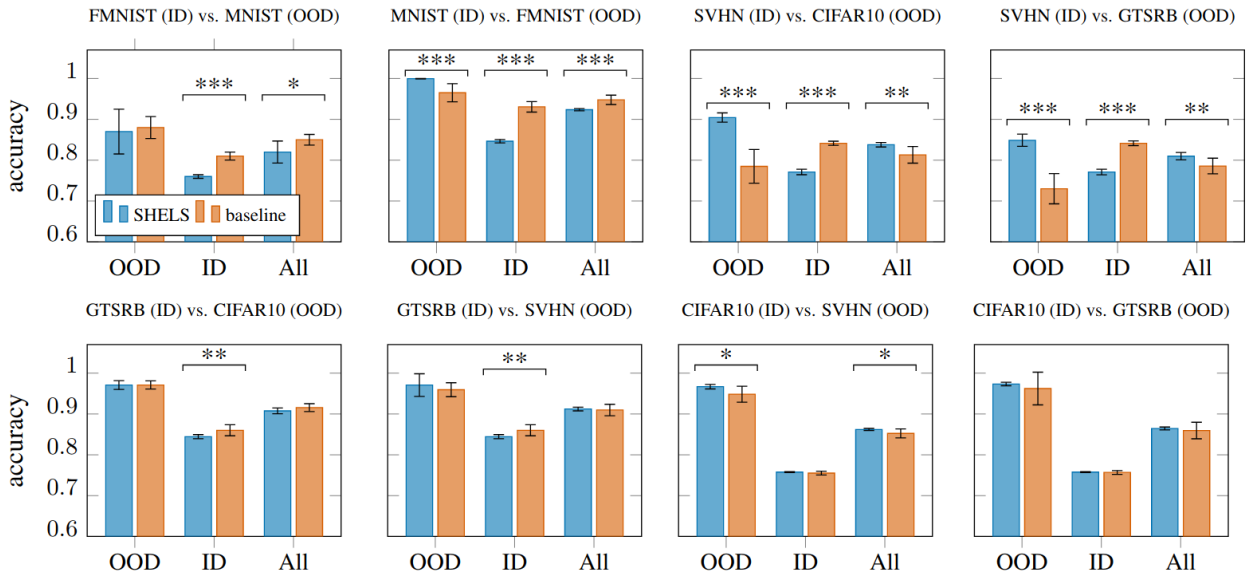


Figure 82: Novelty Detection Across Datasets

combined metric generally shows comparable performance between the methods, exemplifying the nature of the OOD vs. ID tradeoff. As such, while SHELS significantly improves OOD detection at the cost of some ID accuracy, it has comparable overall performance to the state-of-the-art OOD detection baseline on the relatively easier across-datasets experiments.

Detection and classification accuracies for the more challenging within-dataset experiments are shown in Figure 83, with corresponding AUROC measures given in Table 31. The results show a similar OOD vs. ID accuracy trade-off as in the across-datasets experiments, although the differences between SHELS and the baseline become more apparent due to the more difficult within-dataset case. Paired t-tests showed significant improvements in OOD detection using SHELS for four of the five datasets, with no significant difference found for GTSRB. The AUROC measure further supports SHELS's improved OOD detection performance, outperforming the baseline by large margins for many of the datasets. In contrast, paired t-tests showed significant improvements in ID classification accuracy using the baseline approach for all datasets, although SHELS exhibits significant improvements in combined accuracy for four of the five datasets, with no significant difference found for GTSRB. We conclude that our approach's improvements in OOD detection accuracy result in comparable or significant improvements in overall performance compared to the state of the art.

Novelty accommodation We evaluate SHELS's ability to accommodate novel classes through class-incremental continual learning experiments on the MNIST dataset (evaluation on additional datasets is included in our original publication) and compare it with AGS-CL [109], a similar sparsity and weight freezing approach that has shown robust performance, and GDUMB [228], a replay-based method. We also demonstrate the importance of key components of SHELS that enable incremental novelty accommodation, including encouraging group sparsity and penalizing weight updates for novelty accommodation, via ablation studies. We begin by training a classification model to learn SHELS features using the ID train set. For MNIST, the ID data consists of 7 randomly sampled classes and the remaining 3 classes are OOD. The OOD classes are introduced incrementally

Table 30: AUROC of OOD Detection Across Datasets

ID Data	OOD Data	SHELS	Baseline	Sig.
MNIST	FMNIST	**0.99** ± 0.002	0.94 ± 0.03	***
FMNIST	MNIST	**0.88** ± 0.063	0.77 ± 0.064	**
GTSRB	CIFAR10	**0.96** ± 0.01	0.90 ± 0.032	***
GTSRB	SVHN	**0.98** ± 0.005	0.94 ± 0.018	***
SVHN	CIFAR10	0.90 ± 0.007	0.90 ± 0.01	
SVHN	GTSRB	0.88 ± 0.009	**0.89** ± 0.008	*
CIFAR10	GTSRB	**0.93** ± 0.002	0.92 ± 0.025	
CIFAR10	SVHN	**0.97** ± 0.003	0.95 ± 0.05	

Table 31: AUROC of OOD Detection Within Datasets

Dataset	SHELS	Baseline	Sig.
MNIST	**0.96** ± 0.017	0.91 ± 0.062	*
FMNIST	**0.79** ± 0.063	0.74 ± 0.084	
GTSRB	**0.88** ± 0.055	0.81 ± 0.061	*
CIFAR10	**0.70** ± 0.059	0.69 ± 0.067	
SVHN	**0.82** ± 0.023	0.80 ± 0.018	

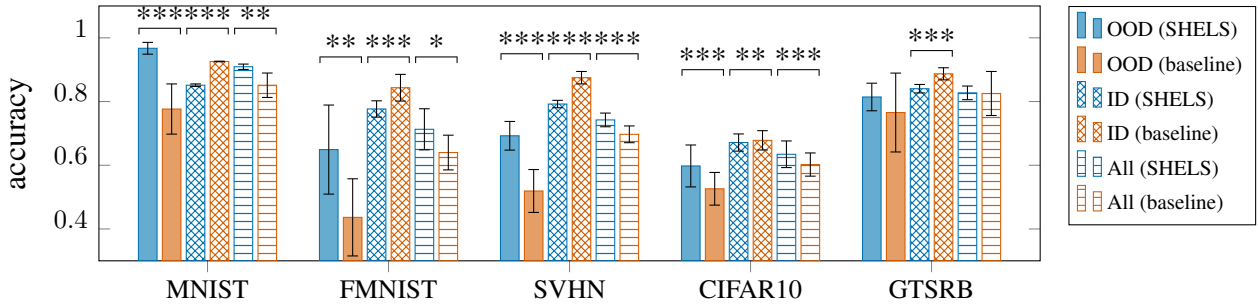


Figure 83: Novelty Detection Within Datasets

in the typical class-incremental continual learning fashion, and our algorithm accommodates them in the SHELS representation.

The class-incremental learning performance of SHELS is shown in the orange performance curves of Figure 84. When a new class is presented to the model, for instance class 8, it experiences a drop in test accuracy depicted by the solid orange curve at epoch 10, as the test set now includes data from the previously unseen class 8. The model then learns class 8, as shown by the dashed novel class test accuracy curve, while preserving performance on the previous seven classes, as evidenced by the upward trend of the overall test accuracy curve. This trend repeats for subsequently introduced novel classes, showing that SHELS learns and accommodates novel classes while mitigating catastrophic forgetting. SHELS significantly outperforms the state-of-the-art AGS-CL method. AGS-CL's performance is closer to SHELS without group sparsity (Ablation 1 below) than to the full SHELS implementation, indicating that SHELS induces sparsity for incorporating new knowledge better than AGS-CL. We also compare our approach to GDUMB, which uses K replay samples balanced across the ID classes. For our experiments, we set K to 1% of the total training data. While GDUMB outperforms SHELS, it is important to note that SHELS does not need to store replay data, which may not be possible in memory-constrained or privacy-sensitive applications. As K increases, GDUMB serves as an upper bound on performance, representing the non-continual learning setting where all data is available to the training algorithm at all times. As the fundamental mechanism of SHELS is complementary to replay, we note that the ideas of SHELS and GDUMB are compatible, and we combine them to yield a new SHELS-GDUMB algorithm in this work's original publication.

Ablation 1: Group Sparsity To show the importance of encouraging group sparsity for novelty

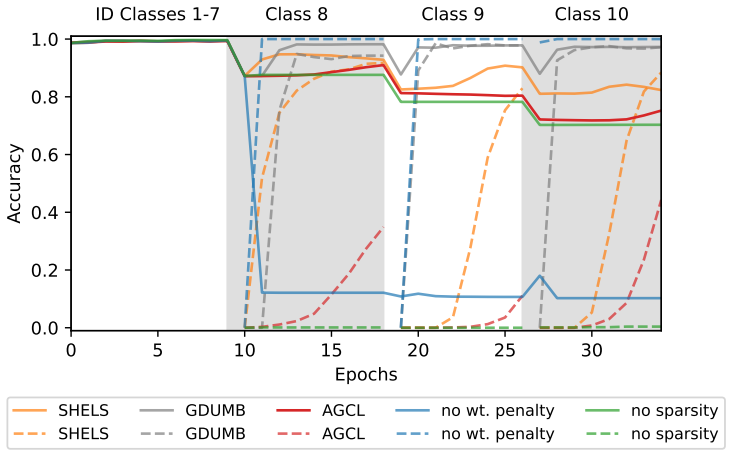


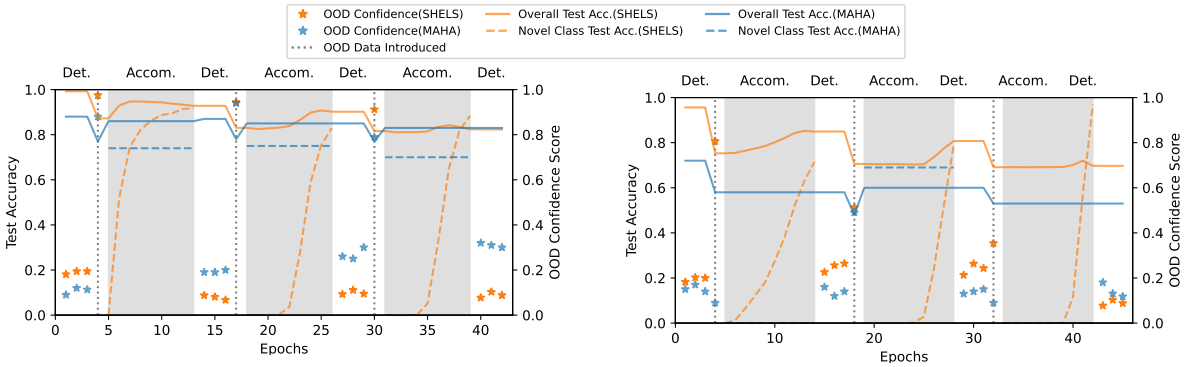
Figure 84: Class Incremental Learning on MNIST

accommodation, we evaluate our method by repeating the same experiment without group sparsity regularization, (*no sparsity* curves in Fig. 84). The regular drop in test accuracy when new classes are introduced and the new class performance staying at zero indicates that the network has no capacity available to accommodate new class information.

Ablation 2: Penalizing Weight Updates To show the effectiveness of penalizing weight updates for novelty accommodation, we evaluate our method without any penalties on weight updates (*no wt. penalty* curves in Fig. 84). While new classes can be learnt successfully, as shown by the new class performance, there is no mechanism to maintain performance on previous tasks, catastrophic forgetting occurs, and test accuracy drops significantly.

The ablations demonstrate the roles of group sparsity and weight update penalization in our method, enabling SHELS to accommodate novel classes in a class-incremental continual learning setting.

Novelty detection and accommodation We demonstrate the performance of SHELS to both detect and accommodate OOD data using the full detection and accommodation pipeline presented at the end of Section 3.4.6. To this end, we conducted a class-incremental continual learning experiment where *class boundaries are not given to the learning agent*. The experiment is designed such that the agent must alternate between novelty detection and accommodation phases, and the decision to switch from detection to accommodation is made by the agent. At each epoch in the detection phase, the agent encounters a random sample of either ID or OOD data. In our experiments, OOD data is always provided at the fourth detection epoch for consistent visualization—note that this consistency has no effect on the agent's ability to detect OOD data. If the agent detects a sample as OOD with a confidence score greater than a preset threshold, it triggers an accommodation phase where it is provided with more of the data from the last detection epoch. The model accommodates this data as a novel class, and once the new class is learnt, it switches back to detection, and the detection/accommodation cycle continues. To the best of our knowledge, this is a novel evaluation paradigm for continual learning without labeled class boundaries. With no methods for this setting in the literature, we provide results from our approach as a case study and compare against a combination of the OOD detection and continual learning components of [229], which we denote MAHA.



(a) MNIST **(b) FMNIST**

Figure 85: Class Incremental Learning Without Boundaries, Alternating between Detection and Accommodation Phases

We use seven randomly sampled classes from the MNIST dataset as ID and the remaining as OOD, and four randomly sampled classes from FMNIST datatset as ID and three out of the remaining six classes as OOD. A performance comparison of our approach with MAHA is shown in Figure 85. For MNIST, we begin by learning a classification model consisting of SHELS features using the ID train set. The model is then deployed and it enters the detection phase. For the first three epochs, it encounters a random sample of test data from the previously seen seven classes, which it correctly identifies as ID as shown by the low OOD confidence scores. At the fourth epoch, the model encounters a sample of OOD class data, correctly identifies it, and switches to the accommodation phase. The newly detected class is accommodated (as shown by the novel class test accuracy curve) while maintaining performance on the previous classes (as shown by the overall test accuracy curve). The model switches back to the detection phase, and the process repeats until the final detection phase, where all ten classes have been accommodated and all data is correctly detected as ID. We repeat the same procedure for FMNIST. For both datasets, SHELS correctly identifies and accommodates all novel classes. In comparison, while MAHA is able to detect and accommodate all novel MNIST classes, it fails to detect two of the novel FMNIST classes, resulting in significantly lower performance when compared to SHELS.

We offer the following explanation for why SHELS outperforms MAHA. For novelty detection, MAHA computes a confidence score using the Mahalanobis distance with respect to the nearest ID class-conditional distribution. While this approach identifies novel classes with features that significantly differ from the ID classes, it fails to identify novel classes with some features similar to the ID classes, which is a frequent case within the FMNIST dataset. SHELS overcomes this drawback by computing the similarity between the high-level activation patterns of the novel class and the exclusive feature sets for each ID class. Since each ID class is represented by a unique set of high-level features, then even if a novel class shares some features with the ID classes, comparing the similarity between these high-level exclusive feature sets still results in effective novelty detection. As for novelty accommodation, MAHA reuses the previously learnt representation, and thus lacks the ability to learn new knowledge. For both the MNIST and FMNIST datasets, MAHA performs poorly when compared to SHELS, since new knowledge needs to be learnt when accommodating new classes. In contrast, due to the sparsity, SHELS is able to learn new knowledge and accommodate new classes more effectively.

The above experiments are analogous to an agent deployed in the open world, where it encounters objects from known classes in its environment. Eventually, the agent will encounter an object of an unknown class, detect that the object is novel, gather additional data to accommodate the new object in its model, and continue to operate in its environment with the updated set of ID classes. Our results validate that the SHELS representation enables integrated novelty detection and accommodation within a single model, and can operate effectively in such an open-world setting.

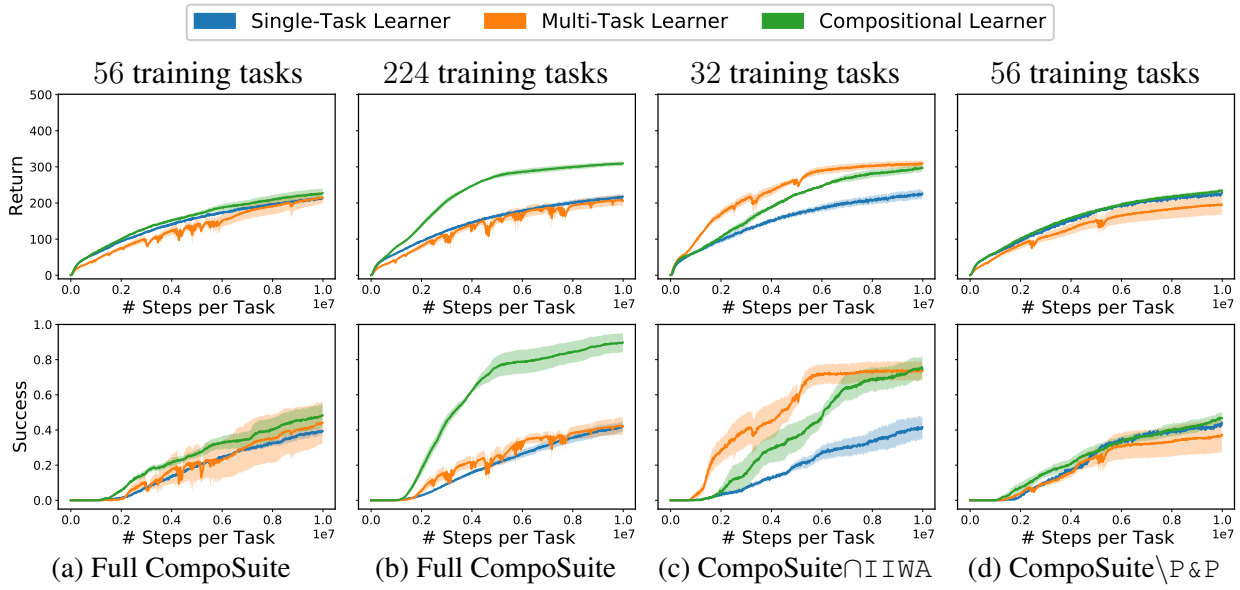
4.5.8 Benchmarking Compositional Multi-task and Lifelong Reinforcement Learning.

The empirical evaluation in this section had two primary objectives. First, to demonstrate that CompoSuite is a useful evaluation benchmark in terms of: 1) existing algorithms making progress toward solving the problems, 2) the tasks exhibiting compositional properties, and 3) existing approaches leaving substantial room for improvement in performance. Second, to provide benchmarking results of existing algorithms for future work to leverage.

Experimental setting The underlying RL algorithm used for all our evaluations was the proximal policy optimization (PPO) [41] implementation in `Spinning Up` [230]. Building upon this base algorithm, we evaluated the following three agents:

- **Single-task** agents that trained on each task individually, without any knowledge-sharing across tasks. Lack of sharing precludes these agents from generalizing to unseen tasks, and so they were only evaluated on training tasks. We also withheld the task descriptor from the observation, as it would appear as a constant to each single-task agent.
- **Multi-task** agents that trained a *shared model* for all tasks, using the task descriptor in the observation to help differentiate between tasks and learn to specialize the policy for each task. Given the need for the multi-task agent to encode multiple policies in a single model, we gave this agent a larger capacity than an individual single-task agent.
- **Compositional** agents that constructed a *different* model for each task from a set of *shared* components. We used a variant of the modular network of [112] that establishes each policy from a set of modules, with one module for each robot, object, obstacle, and objective. The relevant state component from Section 3.4.7 was fed as input to each module, and the task descriptor was used to select the correct modules. Each module was represented by an MLP, whose output was fed to the next module: $\mathsf{obstacle} \rightarrow \mathsf{object} \rightarrow \mathsf{objective} \rightarrow \mathsf{robot}$. For fairness, the *overall* number of parameters across modules was equivalent to that of multi-task agents.

Each agent was evaluated on the full CompoSuite benchmark and on all the suggested smaller-scale and restricted benchmarks. In each of these settings, a subset of the tasks was given to the agents for training, and the agents were evaluated for their speed and final performance over the training tasks. After training, the multi-task and compositional agents were additionally evaluated for their ability to solve unseen tasks without any additional training by leveraging the task descriptors.



(a) Full CompoSuite (b) Full CompoSuite (c) CompoSuite∩IIWA (d) CompoSuite\P&P

Figure 86: Evaluation on Training Tasks From CompoSuite Variants

Evaluation of baselines on the full CompoSuite benchmark We first evaluated the agents on the main CompoSuite benchmark, uniformly sampling tasks for training; learning curves are presented in Figures 86a and 86b. After training for 10 million time steps on each task, the single-task agent had a success rate of around 40%. When the training set was a small portion of the whole set of tasks, the multi-task and compositional agents only slightly improved upon the single-task agent. However, when training on a larger set of tasks, the compositional agent learned much faster, achieving approximately twice as much success. In contrast, multi-task results did not improve with the larger training set. This suggests that the multi-task agent is not appropriately sharing knowledge across tasks, and instead separately allocates capacity in the network to different tasks. As more tasks are seen, capacity is progressively exhausted. Instead, the compositional agent shares components appropriately, and additional training tasks improve the agent's ability to leverage these commonalities. This demonstrates that CompoSuite tasks are indeed compositionally related, and that exploiting these relationships leads to improved performance.

After training, we evaluated the agents on the CompoSuite tasks that they did *not* train on. Intuitively, an agent that correctly decomposes the tasks should achieve high performance on these test tasks by adequately recombining its learned components. Results in the first two columns of Table 32 show that the learners struggled to generalize to unseen tasks when trained on 56 tasks but performed remarkably well when trained on 224 tasks. With the smaller training set, even though the *training* performance was similar for both approaches, the compositional agent achieved substantially worse *zero-shot* performance. This demonstrates that, while the compositional approach can indeed capture the compositional properties of the tasks, this capability requires observing a large portion of the tasks.

One important question is whether the multi-task agent was automatically learning compositional knowledge that allowed it to solve unseen tasks. The alternative explanation would be that the agent instead found similar tasks in the training set and used the policy for those for generalization. We therefore set up a simple experiment, finding the most similar training task to each test task and using

Table 32: CompoSuite Zero-Shot Generalization

	CompoSuite 56 Tasks		CompoSuite 224 Tasks		CompoSuite ∩IIWA		CompoSuite \pick-and-place	
	Multi-task	Comp.	Multi-task	Comp.	Multi-task	Comp.	Multi-task	Comp.
Return	$115.79_{\pm 18.0}$	$64.26_{\pm 7.5}$	$201.74_{\pm 26.9}$	$302.44_{\pm 12.2}$	$232.79_{\pm 17.5}$	$79.85_{\pm 19.2}$	$74.61_{\pm 10.}$	$16.63_{\pm 6.7}$
Success	$0.18_{\pm 0.1}$	$0.08_{\pm 0.0}$	$0.41_{\pm 0.0}$	$0.88_{\pm 0.1}$	$0.49_{\pm 0.1}$	$0.12_{\pm 0.1}$	$0.09_{\pm 0.0}$	$0.01_{\pm 0.0}$

its policy to predict zero-shot performance. Concretely, for every test task $\mathcal{M}_i$ with some zero-shot success, we found the training task $\mathcal{M}_{i'}$ whose policy $\pi_{i'}$ performed best on $\mathcal{M}_i$; this would have been the best policy to choose, and so one would expect the performance of $\pi_{i'}$ to correlate to that of π_i. To reduce computation, we considered only tasks i' that varied in a single element from i. We found that the coefficients of determination between the policies' success rates were very low: $\mathbf{R^2 = 0.19}$ and $\mathbf{R^2 = 0.03}$ for the multi-task and compositional agents, respectively. This shows that the generalization of the multi-task learner was unlikely to come from using trained policies for different tasks, but rather from leveraging the compositional properties of the tasks.

Evaluation of baselines on the smaller-scale CompoSuite∩`IIWA` benchmark Next, we evaluated the three baseline agents on the CompoSuite∩`IIWA` benchmark, in order to 1) propose a computationally cheaper setting to facilitate progress and 2) shed light on the relative difficulty of generalizing across different CompoSuite axes. Learning curves on the training tasks are included in Figure 86c. The relative performance of the compositional agent with respect to the single-task agent was close to that obtained after training on *over 200 tasks* for the full CompoSuite, demonstrating that the agent is capable of discovering the compositional structure of this reduced benchmark with far fewer training tasks. On the other hand, the multi-task agent performed noticeably better in CompoSuite∩`IIWA` and CompoSuite∩`pick-and-place`. This provides evidence that these two axes (robot and objective) are harder to generalize across, as one would intuitively expect.

We further assessed the performance of the agents on the unseen `IIWA` tasks, and report the results in the third column of Table 32. The multi-task agent achieved notably high performance but the compositional agent was incapable of generalizing. The poor generalization of the compositional agent was likely due to the small number of training tasks: since the agent only trains each module on the subset of tasks that shares that module, each parameter was trained on a small number of tasks which was insufficient for zero-shot generalization.

Evaluation of baselines on the restricted CompoSuite\`pick-and-place` benchmark The results presented so far consider a relatively simple compositional problem: the agent is trained on multiple combinations of all components, and is expected to generalize to new combinations. These previous results already expose the shortcomings of existing approaches in the compositional setting. However, future approaches that solve these simple settings would still fall short from achieving the full compositional capabilities we expect from them. We would hope that agents could learn components that generalize to unseen tasks even if these components are only seen in one single combination. To study this setting, we evaluated the three agents on the restricted CompoSuite\`pick-and-place` benchmark. Results on the training tasks, which include exactly one `pick-and-place` task, are shown in Figure 86d. Performance was close to that of the full benchmark using 56 tasks, because the training distributions are similar: there are 55 combinations of 15 components (plus one `pick-and-place` task), compared to 56 combinations of 16 components. The fourth column of Table 32 summarizes the `pick-and-place` zero-shot

Table 33: Zero-Shot Generalization (Success Rate) for the Multi-Task Agent on CompoSuite\pick-and-place, Separated by Tasks That Share (or Not) Each Element with the Trained pick-and-place Task

Element	Trained	Untrained
robot	$0.30_{\pm 0.10}$	$0.03_{\pm 0.02}$
object	$0.14_{\pm 0.04}$	$0.08_{\pm 0.04}$
obstacle	$0.14_{\pm 0.04}$	$0.08_{\pm 0.03}$

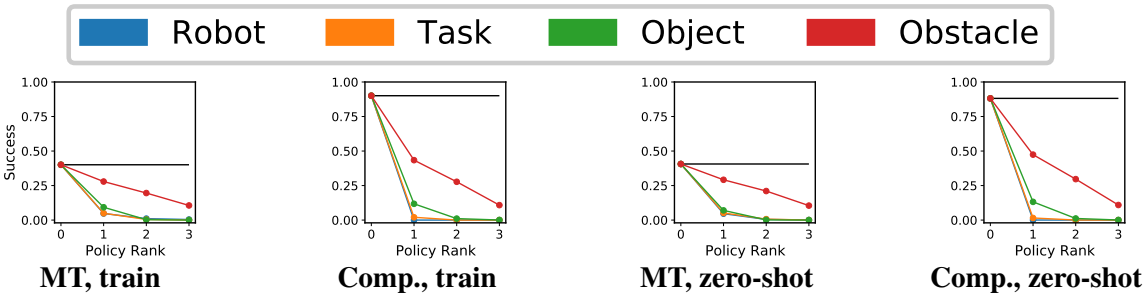


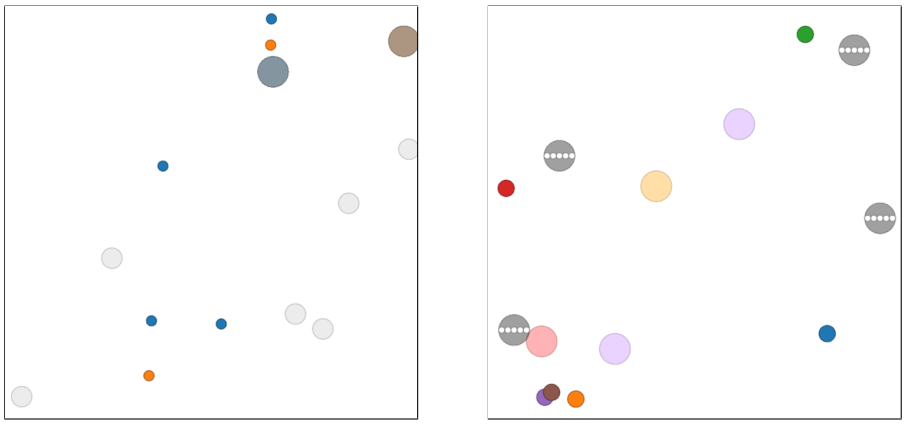
Figure 87: Performance Given Incorrect Task Descriptors, With a Single Changed Component

performance of the agents on this restricted setting. Both agents failed to generalize to the unseen `pick-and-place` tasks. Table 33 shows that the small amount of generalization the multi-task agent achieved was almost entirely on tasks from the same robot arm that was used in the single `pick-and-place` training task.

We now shift focus to verifying two important properties of CompoSuite: that the large majority of tasks are learnable by current RL mechanisms, and that the tasks are not only compositional, but also highly varied.

Learnability of tasks If CompoSuite tasks were unsolvable by current RL methods, that would conflate the difficulty of compositional reasoning with the difficulty of solving RL tasks. To validate that this is not the case, for every task, we found the best performing agent across all those trained in Figure 86 (taking the maximum across experiments *and* random seeds). For any task with a score of 0, we have no evidence that the task is learnable, because no agents solved it to any extent. The result of this computation yielded that only one task received a score of 0, indicating that it *may* be unlearnable; this demonstrates that CompoSuite tasks are attainable for current RL methods.

Diversity of tasks Another valid concern is that it might be possible for the agent to solve multiple tasks with a single policy if the tasks are very similar, implying that compositional reasoning is not necessary for generalization. To verify that this is not the case in CompoSuite, after training the multi-task and compositional agents over 224 tasks, we evaluated their performance if they were given the *incorrect* task descriptor. In particular, for a given task $\mathcal{M}_i$, we tested the performance of the agent on task $\mathcal{M}_i$ if given the descriptor for all tasks $\mathcal{M}_{i'}$ that varied in a single component from $\mathcal{M}_i$ (i.e., the tasks most similar to i). We sorted the performances with these incorrect descriptors separately for each axis, finding the rank of each incorrect component (e.g., the rank-two robot for task $\mathcal{M}_i$ is the robot that achieved the second-best performance when used as the descriptor for task $\mathcal{M}_i$), and averaged the sorted performances. The results, summarized in Figure 87, show that using the incorrect robot, objective or object descriptor consistently leads to substantially degraded



(a) Cooperative Treasure Collection environment

(b) Rover-Tower environment

Figure 88: MAAC - Environments

performance, particularly for the compositional agent; this means that the agents are specializing to each of the components. Using the incorrect obstacle descriptor has a much smaller impact, particularly for the multi-task agent, which suggests that the multi-task agent learns a policy that is somewhat agnostic to the obstacles. We conclude that CompoSuite cannot be solved without specializing the policies to each task. Additionally, varying the robot arm causes a drastic drop in performance, demonstrating that solving tasks with varied robots is a challenging problem, yet existing benchmarks are limited to a single robot arm.

4.5.9 Multi-actor-attention-critic for multi-agent reinforcement learning.

Setup We construct two environments that test various capabilities of our approach (MAAC) and baselines. We investigate in two main directions. First, we study the scalability of different methods as the number of agents grows. We hypothesize that the current approach of concatenating all agents' observations (often used as a global state to be shared among agents) and actions in order to centralize critics does not scale well. To this end, we implement a cooperative environment, Cooperative Treasure Collection, with partially shared rewards where we can vary the total number of agents without significantly changing the difficulty of the task. As such, we can evaluate our approach's ability to scale.

Secondly, we want to evaluate each method's ability to attend to information relevant to rewards, especially when the relevance (to rewards) can dynamically change during an episode. This scenario is analogous to real-life tasks such as the soccer example presented earlier. To this end, we implement a Rover-Tower task environment where randomly paired agents communicate information and coordinate.

Finally, we test on the Cooperative Navigation task proposed by [18] in order to demonstrate the general effectiveness of our method on a benchmark multi-agent task.

All environments are implemented in the multi-agent particle environment framework[12] introduced

[12]https://github.com/openai/multiagent-particle-envs

Table 34: MAAC - Baseline Comparison

	Base Algorithm	How to incorporate other agents	Number of Critics	Multi-task Learning of Critics	Multi-Agent Advantage
MAAC (ours)	SAC‡	Attention	N	✓	✓
MAAC (Uniform) (ours)	SAC	Uniform Atttention	N	✓	✓
COMA*	Actor-Critic (On-Policy)	Global State + Action Concatenation	1		✓
MADDPG†	DDPG**	Observation and Action Concatenation	N		
COMA+SAC	SAC	Global State + Action Concatenation	1		✓
MADDPG+SAC	SAC	Observation and Action Concatenation	N		✓

Heading Explanation *How to incorporate other agents*: method by which the centralized critic(s) incorporates observations and/or actions from other agents (MADDPG: concatenating all information together. COMA: a global state instead of concatenating observations; however, when the global state is not available, all observations must be included.) *Number of Critics*: number of separate networks used for predicting Q_i for all N agents. *Multi-task Learning of Critics*: all agents' estimates of Q_i share information in intermediate layers, benefiting from multi-task learning. *Multi-Agent Advantage*: cf. Sec 3.2 for details.

Citations: *[19], †[18], ‡[117], **[40]

by [231], and extended by [18]. We found this framework useful for creating environments involving complex interaction between agents, while keeping the control and perception problems simple, as we are primarily interested in addressing agent interaction. To further simplify the control problem, we use discrete action spaces, allowing agents to move up, down, left, right, or stay; however, the agents may not immediately move exactly in the specified direction, as the task framework incorporates a basic physics engine where agents' momentums are taken into account. Figure 88 illustrates the two environments we introduce.

Cooperative Treasure Collection The cooperative environment in Figure 88a) involves 8 total agents, 6 of which are "treasure hunters" and 2 of which are "treasure banks", which each correspond to a different color of treasure. The role of the hunters is to collect the treasure of any color, which re-spawn randomly upon being collected (with a total of 6), and then "deposit" the treasure into the correctly colored "bank". The role of each bank is to simply gather as much treasure as possible from the hunters. All agents are able to see each others' positions with respect to their own. Hunters receive a global reward for the successful collection of treasure and all agents receive a global reward for the depositing of treasure. Hunters are additionally penalized for colliding with each other. As such, the task contains a mixture of shared and individual rewards and requires different "modes of attention" which depend on the agent's state and other agents' potential for affecting its rewards.

Rover-Tower The environment in Figure 88b involves 8 total agents, 4 of which are "rovers" and another 4 which are "towers". At each episode, rovers and towers are randomly paired. The pair is negatively rewarded by the distance of the rover to its goal. The task can be thought of as a navigation task on an alien planet with limited infrastructure and low visibility. The rovers are unable to see in their surroundings and must rely on communication from the towers, which are able to locate the rovers as well as their destinations and can send one of five discrete communication

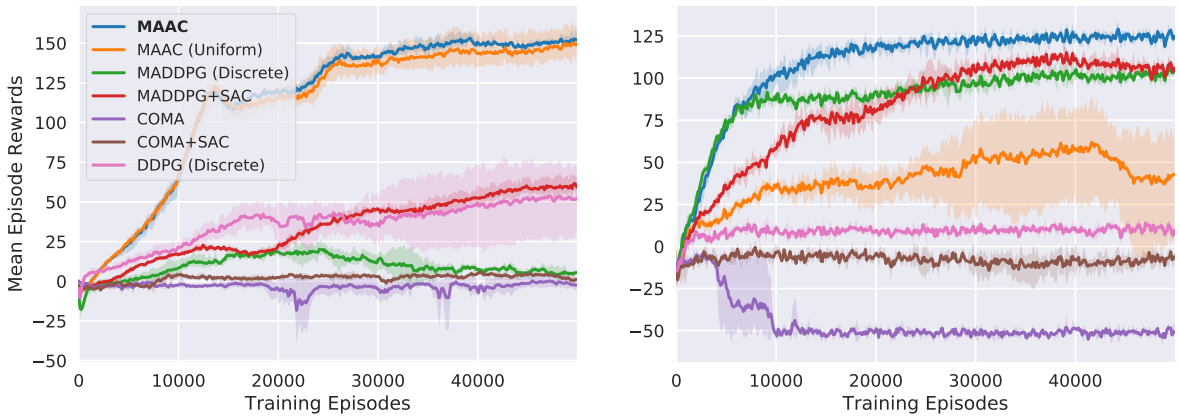


Figure 89: MAAC - Results (Left) Average Rewards on Cooperative Treasure Collection (Right) Average Rewards on Rover-Tower

Table 35: Average Rewards per Episode on Cooperative Navigation

MAAC	MAAC (Uniform)	MADDPG+SAC	COMA+SAC
-1.74 $\pm$ 0.05	-1.76 $\pm$ 0.05	-2.09 $\pm$ 0.12	-1.89 $\pm$ 0.07

messages to their paired rover. Note that communication is highly restricted and different from centralized policy approaches [134], which allow for free transfer of continuous information among policies. In our setup, the communication is integrated into the environment (in the tower's action space and the rover's observation space), rather than being explicitly part of the model, and is limited to a few discrete signals.

Baselines We compare to two recently proposed approaches for centralized training of decentralized policies: MADDPG [18] and COMA [19], as well as a single-agent RL approach, DDPG, trained separately for each agent.

As both DDPG and MADDPG require differentiable policies, and the standard parametrization of discrete policies is not differentiable, we use the Gumbel-Softmax reparametrization trick [232]. We will refer to these modified versions as MADDPG (Discrete) and DDPG (Discrete). Our method uses Soft Actor-Critic to optimize. Thus, we additionally implement MADDPG and COMA with Soft Actor-Critic for the sake of fair comparison, referred to as MADDPG+SAC and COMA+SAC.

We also consider an ablated version of our model as a variant of our approach. In this model, we use uniform attention by fixing the attention weight α_j (Eq. 52) to be $1/(N-1)$. This restriction prevents the model from focusing its attention on specific agents.

All methods are implemented such that their approximate total number of parameters (across agents) are equal to our method, and each model is trained with 6 random seeds each. Hyperparameters for each underlying algorithm are tuned based on performance and kept constant across all variants of critic architectures for that algorithm. A thorough comparison of all baselines is summarized in Table 34.

Results and Analysis Figure 89 illustrates the average rewards per episode attained by various methods on our two environments, and Table 35 displays the results on Cooperative Navigation [18].

Our proposed approach (MAAC) is competitive when compared to other methods. We analyze in detail in below.

Impact of Rewards and Required Attention Uniform attention is competitive with our approach in the Cooperative Treasure Collection (CTC) and Cooperative Navigation (CN) environments, but not in Rover-Tower. On the other hand, both MADDPG (Discrete) and MADDPG+SAC perform well on Rover-Tower, though they do not on CTC. Both variants of COMA do not fare well in CTC and Rover-Tower, though COMA+SAC does reasonably well in CN. DDPG, arguably a weaker baseline, performs surprisingly well in CTC, but does poorly in Rover-Tower.

In CTC and CN, the rewards are shared across agents thus an agent's critic does not need to focus on information from specific agents in order to calculate its expected rewards. Moreover, each agent's local observation provides enough information to make a decent prediction of its expected rewards. This might explain why MAAC (Uniform) which attends to other agents equally, and DDPG (unaware of other agents) perform above expectations.

On the other hand, rewards in the Rover-Tower environment for a specific agent are tied to another single agent's observations. This environment exemplifies a class of scenarios where dynamic attention can be beneficial: when subgroups of agents are interacting and performing coordinated tasks with separate rewards, but the groups do not remain static. This explains why MAAC (Uniform) performs poorly and DDPG completely breaks down, as knowing information from another specific agent is *crucial* in predicting expected rewards.

COMA uses a single centralized network for predicting Q-values for all agents with separate forward passes. Thus, this approach may perform best in environments with global rewards and agents with similar action spaces such as Cooperative Navigation, where we see that COMA+SAC performs well. On the other hand, the environments we introduce contain agents with differing roles (and non-global rewards in the case of Rover-Tower). Thus both variants of COMA do not fare well.

MADDPG (and its Soft Actor-Critic variant) perform well on RT; however, we suspect their low performance in CTC is due to this environment's relatively large observation spaces for all agents, as the MADDPG critic concatenates observations for all agents into a single input vector for each agent's critic. Our next experiments confirms this hypothesis.

Scalability In Table 36 we compare the average rewards attained by our approach and the next best performing baseline (MADDPG+SAC) on the CTC task (normalized by the range of rewards attained in the environment, as differing the number of agents changes the nature of rewards in this environment). We show that the improvement of our approach over MADDPG+SAC grows with respect to the number of agents. As suspected, MADDPG-like critics use all information non-selectively, while our approach can learn which agents to pay more attention through the attention mechanism and compress that information into a constant-sized vector. Thus, our approach scales better when the number of agents increases. In future research we will continue to improve the scalability when the number of agents further increases by sharing policies among agents, and performing attention on sub-groups (of agents).

In Figure 90 we compare the average rewards per episode on the Rover-Tower task. We can compare rewards directly on this task since each rover-tower pair can attain the same scale of rewards regardless of how many other agents are present. Even though MADDPG performed well on the 8 agent version of the task (shown in Fig. 89), we find that this performance does not scale. Meanwhile, the performance of MAAC does not deteriorate as agents are added.

Table 36: Cooperative Treasure Collection Scalability

# Agents	4	8	12
% Improvement	17	98	208

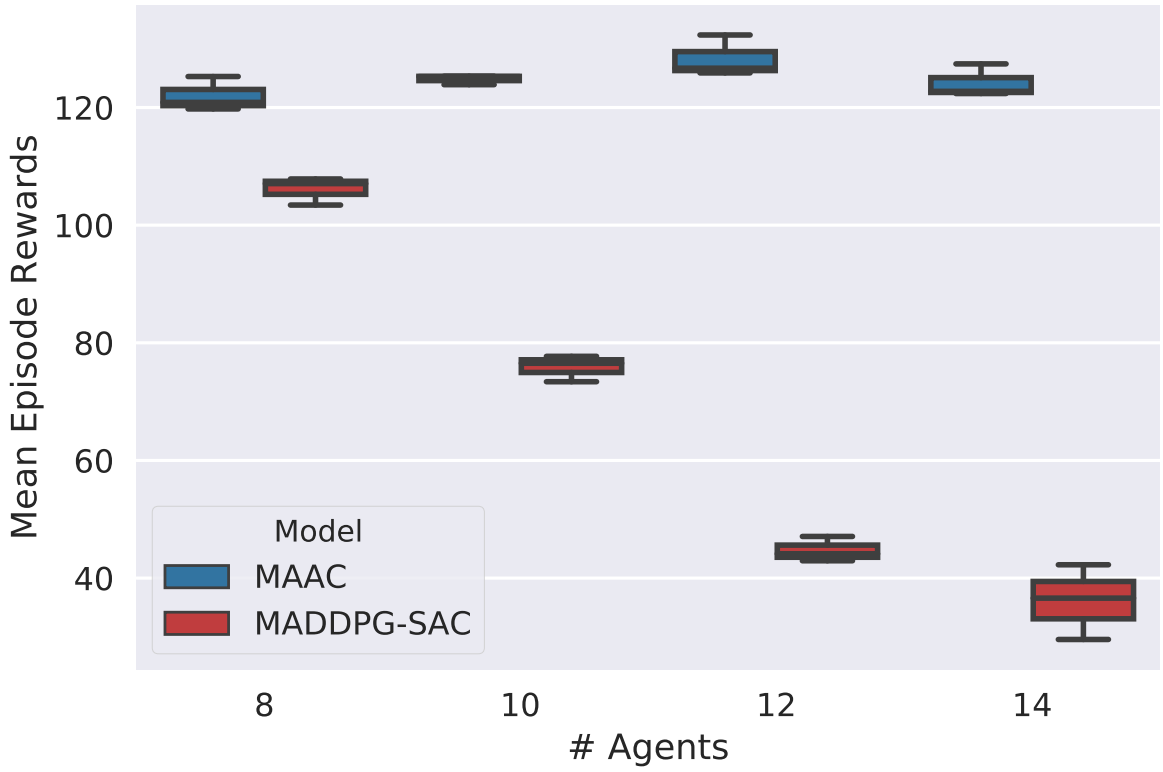


Figure 90: Scalability in the Rover-Tower Task

As a future direction, we are creating more complicated environments where each agent needs to cope with a large group of agents where selective attention is needed. This naturally models real-life scenarios that multiple agents are organized in clusters/sub-societies (school, work, family, etc) where the agent needs to interact with a small number of agents from many groups. We anticipate that in such complicated scenarios, our approach, combined with some advantages exhibited by other approaches will perform well.

Visualizing Attention In order to inspect how the attention mechanism is working on a more fine-grained level, we visualize the attention weights for one of the rovers in Rover-Tower (Fig. 91), while fixing the tower that said rover is paired to. In this plot, we ignore the weights over other rovers for simplicity since these are always near zero. We find that the rover learns to strongly attend to the tower that it is paired with, without any explicit supervision signal to do so. The model implicitly learns which agent is most relevant to estimating the rover's expected future returns, and said agent can change dynamically without affecting the performance of the algorithm.

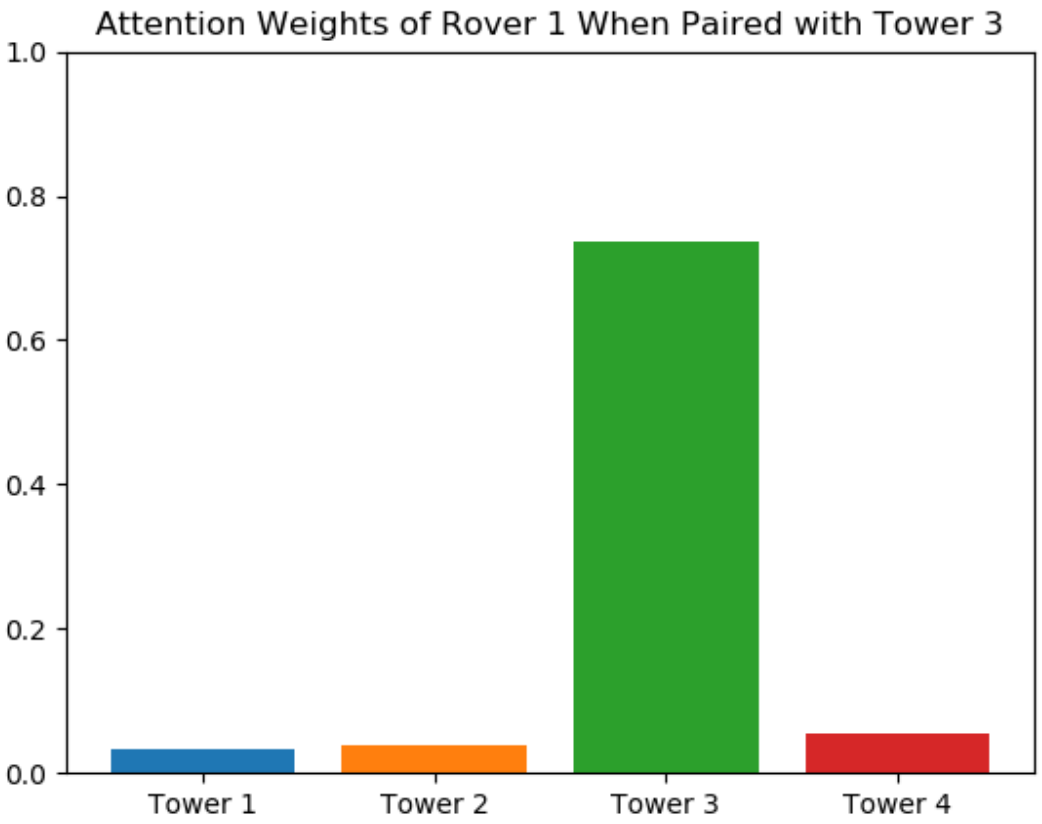
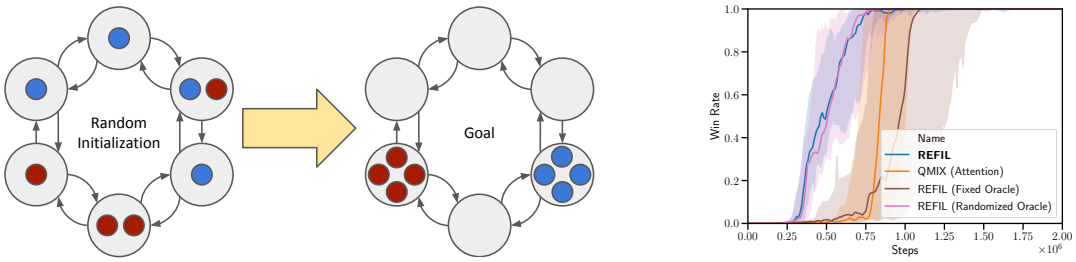


Figure 91: Attention Visualization

4.5.10 Randomized entity-wise factorization (REFIL) for multi-agent reinforcement learning.

In our experiments, we aim to answer the following questions: 1) Are randomized counterfactuals an efficient means for leveraging common patterns? 2) Does our approach improve generalization in a multi-task setting? 3) Is training as an auxiliary objective justified? We begin with experiments in a simple domain we construct such that agents' decisions rely only on a *known subset* of all entities, so we can compare our approach to those that use this domain knowledge. Then, we move on to testing on complex StarCraft micromanagement tasks to demonstrate our method's ability to scale to complex domains.

Group Matching Game In order to answer our first question, we construct a group matching game, pictured in Figure 92a, where each agent only needs to consider a subset of other agents to act effectively and we know that subset as ground truth (unlike in more complex domains such as StarCraft). Agents (of which there are n_a) are randomly placed in one of n_c cells and assigned to one of n_g groups (represented by the different colors) at the start of each episode. Each unique group assignment corresponds to a task. Agents can choose from three actions: move clockwise, stay, and move counter-clockwise. Their ultimate goal is to be located in the same cell as the rest of



(a) Game Visualization **(b) Win Rate over Time**

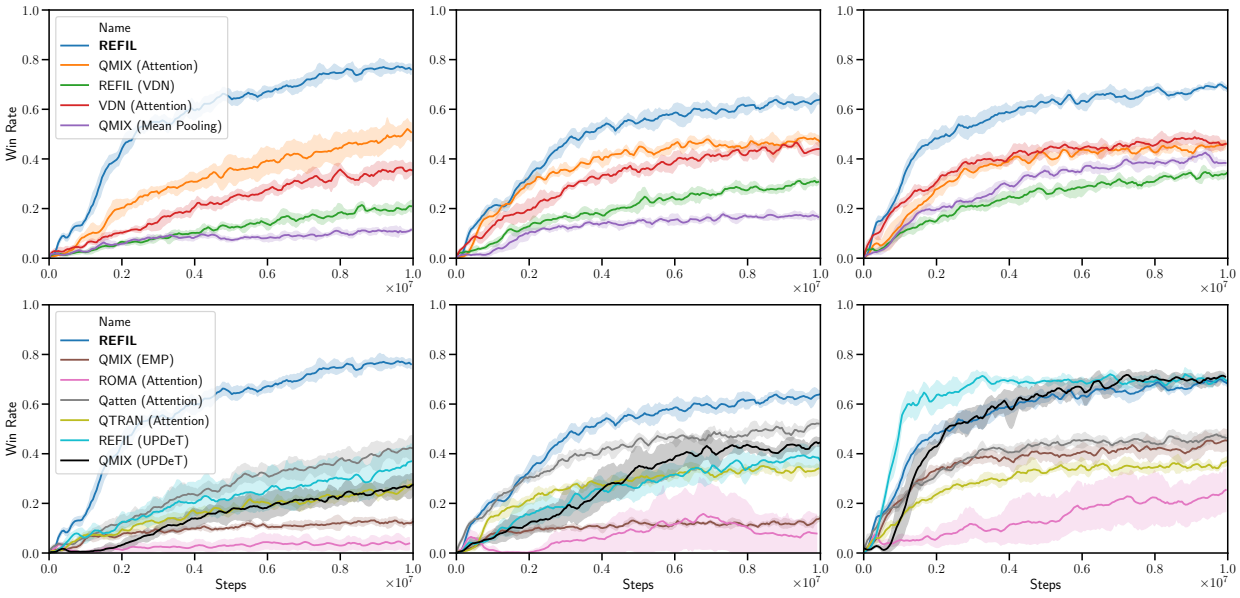
Figure 92: REFIL Group Matching Game Results

their group members, at which point an episode ends. There is no restriction on which cell agents form a group in (e.g., both groups can form in the same cell). All agents share a reward of 2.5 when any group is completed (and an equivalent penalty for a formed group breaking) as well as a penalty of -0.1 for each time step in order to encourage agents to solve the task as quickly as possible. Agents' entity-state descriptions s^e include the cell that the agent is currently occupying as well as the group it belongs to (both one-hot encoded), and the task is fully-observable. Notably, agents can act optimally while only considering a subset of observed entities.

Ground-truth knowledge of relevant entities enables us to disentangle two aspects of our approach: the use of entity-wise factorization in general and specifically using randomly selected factors. We would like to answer the question: does our method rely on sampling the "right" groups of entities (i.e., those with no interactions between them), or is the randomness of our method a feature that promotes generalization? We construct two approaches that use this knowledge to build factoring masks M_I and M_O that are used in place of randomly sampled groups (otherwise the methods are identical to **REFIL**). **REFIL** *(Fixed Oracle)* directly uses the ground truth group assignments (different for each task) to build masks. **REFIL** *(Randomized Oracle)* randomly samples sub-groups from the ground truth groups only, rather than from all possible entities. We additionally train **REFIL** and *QMIX (Attention)* (i.e., **REFIL** with no auxiliary loss).

Figure 92b shows that using domain knowledge does not significantly improve performance in this domain (*QMIX (Attention)* vs. **REFIL** *(Fixed Oracle)*). In fact our *randomized* factorization approach outperforms the use of domain knowledge. The randomization in **REFIL** therefore appears to be crucial. Our hypothesis is that randomization of sub-group factors enables better knowledge sharing across tasks. For example, the situation where two agents from the same group are located in adjacent cells occurs within *all* possible group assignments. When sampling randomly, our approach occasionally samples these two agents alone in their own group. Even if the rest of the context in a given episode has never been seen before, as long as this sub-scenario has been seen, the model has some indication of the value associated with each action. Even when restricting the set of entities to form sub-groups with those that we know can be relevant to each agent (**REFIL** *(Randomized Oracle)*) we find that performance does not significantly improve. These results suggest that randomized sub-group formation for **REFIL** is a viable strategy, and the main benefit of our approach is to promote generalization across tasks by breaking value function predictions into reusable components, even when the sampled sub-groups are not completely independent.

STARCRAFT We next test on the StarCraft multi-agent challenge (SMAC) [233]. The tasks in SMAC involve micromanagement of units in order to defeat a set of enemy units in battle.



(a) 3-8sz **(b) 3-8csz** **(c) 3-8MMM**

Figure 93: REFIL StarCraft Average Win Rate Across Tasks Over Time

Specifically, we consider a multi-task setting where we train our models simultaneously on tasks with variable types and quantities of agents. We hypothesize that our approach is especially beneficial in this setting, as it should encourage models to learn utilities for common patterns and generalize to more diverse settings as a result. The dynamic setting involves minor modifications to SMAC but we change the environment as little as possible to maintain the challenging nature of the tasks. In the standard version of SMAC, both state and action spaces depend on a fixed number of agents and enemies, so our modifications, alleviate these problems.

In our tests we evaluate on three settings we call 3-8sz, 3-8csz, and 3-8MMM. 3-8sz pits symmetrical teams of between 3 and 8 agents against each other where the agents are a combination of Zealots and Stalkers (inspired by the 2s3z and 3s5z tasks in the original SMAC), resulting in 39 unique tasks. 3-8csz pits symmetrical teams of between 0 and 2 Colossi and 3 to 6 Stalkers/Zealots against each other (inspired by 1c3s5z), resulting in 66 tasks. 3-8MMM pits symmetrical teams of between 0 and 2 Medics and 3 to 6 Marines/Marauders against each other (inspired by MMM and MMM2, again resulting in 66 tasks).

Ablations and Baselines We introduce several ablations of our method, as well as adaptations of existing methods to handle variable sized inputs. These comparisons are summarized in Table 37. *QMIX (Attention)* is our method without the auxiliary loss. **REFIL** (VDN) is our approach using summation to combine all factors as in VDN rather than a non-linear monotonic mixing network. *VDN (Attention)* does not include the auxiliary loss and uses summation for factor mixing. *QMIX (Mean Pooling)* is *QMIX (Attention)* with attention layers replaced by mean pooling. We also test max pooling but find the performance to be marginally worse than mean pooling. Importantly, for pooling layers we add entity-wise linear transformations prior to the pooling operations such that the total number of parameters is comparable to attention layers.

For baselines we consider some follow-up works to QMIX that improve the mixing network's expressivity: QTRAN [237] and Qatten [236]. We also compare to a method that builds on QMIX

Table 37: REFIL Baseline Comparison

Name	Imagined Learning	Model	Base Algorithm
REFIL	✓	MHA[1]	QMIX[2]
QMIX (Attention)		MHA	QMIX
REFIL (VDN)	✓	MHA	VDN[3]
VDN (Attention)		MHA	VDN
QMIX (Max Pooling)		Max-Pool	QMIX
QMIX (EMP)		EMP[4]	QMIX
ROMA (Attention)		MHA	ROMA[5]
Qatten (Attention)		MHA	Qatten[6]
QTRAN (Attention)		MHA	QTRAN[7]
REFIL (UPDeT)	✓	UPDeT[8]	QMIX
QMIX (UPDeT)		UPDeT	QMIX

[1]: [120] [2]: [130] [3]: [129]
[4]: [234] [5]: [235] [6]: [236]
[7]: [237] [8]: [238]

by attempting to learn dynamic roles that depend on the context each agent observes: ROMA [235]. We additionally consider an alternative mechanism for aggregating information across variable sets of entities, known as Entity Message Passing (EMP) [234]. We specifically use the restricted communication setting where agents can only communicate with agents they observe, and we set the number of message passing steps to three. Finally, we consider the UPDeT architecture [238], a recent work that also targets the multi-task MARL setting. UPDeT utilizes domain knowledge of the environment to map entities to the specific actions that they correspond to. We train UPDeT with QMIX as well as **REFIL**. For all approaches designed for the standard single-task SMAC setting, we extend them with the same multi-head attention architecture that our approach uses.

Ablation Results Our results on challenges in multi-task StarCraft settings can be found in Figure 93. Tasks are sampled uniformly at each episode, so the curves represent average win rate across all tasks. We find that **REFIL** outperforms all ablations consistently in these settings. **REFIL** *(VDN)* performs much worse than our approach and *VDN (Attention)*, highlighting the importance of the mixing network handling contextual dependencies between entity partitions. Since the trajectory of a subset of entities can play out differently based on the surrounding context, it is important for our factorization approach to recognize and adjust for these situations. The use of mean-pooling in place of attention also performs poorly, indicating that attention is valuable for aggregating information from variable length sets of entities.

Baseline Results We find that algorithms designed to improve on QMIX for the single task MARL setting (ROMA, Qatten, QTRAN), naively applied to the multi-task setting, do not see the same improvements. **REFIL**, on the other hand, consistently outperforms other methods, highlighting the unique challenge of learning in multi-task settings. In Figure 94 (top right) we investigate the performance of **REFIL** compared to the two next best methods in the 3-8sz setting on a task-by-task basis. We evaluate each method on each task individually and rank the tasks by performance,

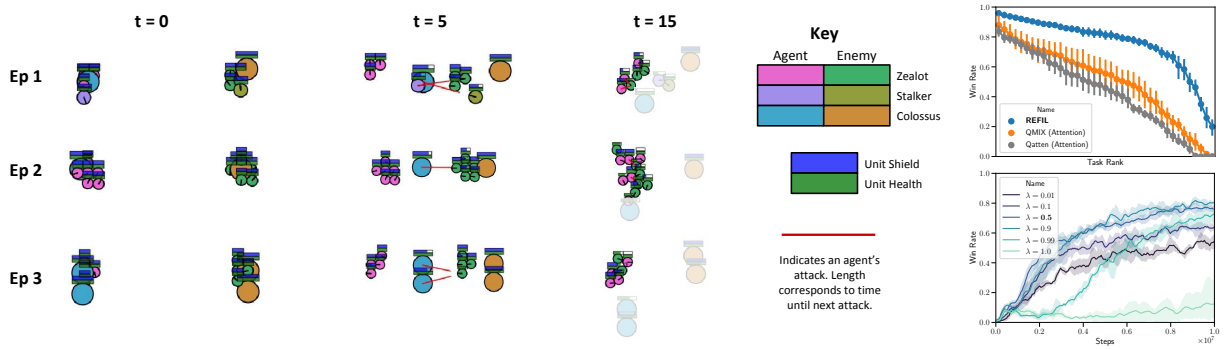


Figure 94: REFIL Environment Visualization (left), Generalization Results (top right), and Varying Value of λ (bottom right)

plotting from left to right. We find that the performance gain of **REFIL** comes from generalizing performance across a wider range of tasks, hence the reduced rate of decay in task performance from best to worst.

The entity aggregation method of EMP underperforms relative to the MHA module that we use. UPDeT is a related work that focuses on designing an architecture compatible with multiple tasks and variable entities and action spaces by utilizing domain knowledge to map entities to their corresponding actions. Despite adding this domain knowledge, QMIX (UPDeT) surprisingly underperforms in 2 of 3 settings, while performing similarly to **REFIL** on 3-8MMM; however, since UPDeT is an attention-based architecture, it is amenable to our proposed auxiliary training scheme. We find applying random factorization to QMIX (UPDeT) improves its performance further in 3-8MMM as well as in 3-8sz. In the case of 3-8MMM, where the asmyptotic win rate of **REFIL** (UPDeT) and QMIX (UPDeT) are similar, we find that **REFIL** (UPDeT) wins on average in 22% fewer time steps by targeting enemy Medivacs, a unit capable of healing its teammates. Targeting of Medivacs is an example of a common pattern that emerges across tasks which **REFIL** is able to leverage.

Role of Auxiliary Objective In order to understand the role of training as an auxiliary objective (rather than entirely replacing the objective) we vary the value of λ to interpolate between two modes: $\lambda = 0$ is simply *QMIX (Attention)*, while $\lambda = 1$ trains exclusively with random factorization. Our results on 3-8sz (Fig. 94 (bottom right)) show that, similar to regularization methods such as Dropout [239], there is a sweet spot where performance is maximized before collapsing catastrophically. Training exclusively with random factorization does not learn anything significant. This failure is likely due to the fact that we use the full context in our targets for learning with imagined scenarios as well as when executing our policies, so we still need to learn with it in training.

Qualitative Example of Common Pattern Finally, we visualize an example of the sort of common patterns that **REFIL** is able to leverage (Fig. 94 (left)). Zealots (the only melee unit present) are weak to Colossi, so they learn to hang back and let other units engage first. Then, they jump in and intercept the enemy Zealots while all other enemy units are preoccupied, leading to a common pattern of a Zealot vs. Zealot skirmish (highlighted at t=15). **REFIL** enables behaviors learned in these types of sub-groups to be applied more effectively across all tasks. By sampling groups from all entities randomly, we will occasionally end up with sub-groups that include only Zealots, and the value function predictions learned in these sub-groups can be applied not only to the task at hand, but to any task where a similar pattern emerges.

4.5.11 Solving stochastic traveling purchaser problem (STPP) for scavenger hunt service robot.

To compare the algorithms proposed in 3.4.10, we implemented all seven algorithms along with a simulator which allows us to simulate STPPs specified by a graph world, set of objects, and distribution model for the objects. Each simulated episode of scavenger hunt randomizes the object arrangement according to the distribution model. [13] STPPs are randomly generated according to the following rules.

1. Each STPP contains 4 objects.
2. An object may be located at between 1 and 3 locations, with the probability of appearing at each location partitioned randomly between locations.
3. Node locations are generated uniformly at random in an environment of size m by m, where m is 100 times the number of nodes, and the edge distances are calculated as Euclidean distances based on the node locations.

To evaluate the performances of the algorithms, we generated 10 random 8-node environments. The algorithms are evaluated in each environment for 100 i.i.d episodes. Note that, for the DQN algorithm, 10 different DQN policies were trained and tested on 10 environments separately. We tested two versions of the DQN algorithm. The first, *DQN*, received the distribution as part of its observation, along with the robot's location, and the list of objects that remain to be found, while the second, *DQN+Map* received both the distribution and the map edge costs as part of the observation.

The average travel distance by each algorithm to the average travel distance by the offline optimal algorithm is reported in Figure 95. The results indicate that the DQN algorithm can outperform all other heuristics, even without knowing the edge costs. When provided with information about edge costs it can perform as well as Bayesian search with no statistical difference in performance between DQN+Map and Bayesian search with $p - value > 0.05$. Moreover, the difference between DQN and the next best heuristic, i.e. Prob-Prox, is statistically significant with $p - value = 0.006$ in a paired two sample T-test.

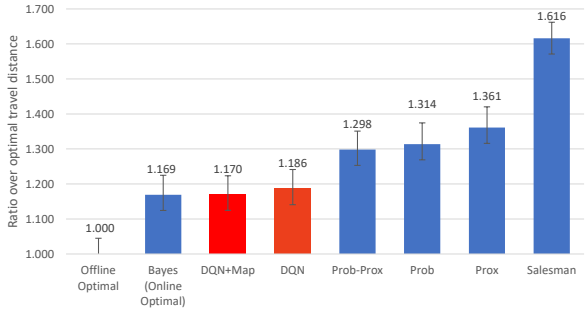


Figure 95: Comparison of Algorithms on 10 Environments. The DQN Algorithms were Trained on Each Environment Separately and Outperforms All Other Heuristics.

[13] The simulator code is available at: https://github.com/utexas-bwi/scavenger_hunt_api/tree/master/bwi_scavenger

We also examined the dependence of performances w.r.t to number of nodes in the environments. We ran sets of trials on eight graph sizes of 3 through 10 nodes. For each graph size, each algorithm completed 100 trials for each of the 100 i.i.d episodes, for a total of 80,000 trials per algorithm (with the exception of exhaustive Bayesian search, which could not be run on maps of sizes 9 or 10 nodes in reasonable time due to its computational complexity. We estimate it would take approximately 8 days to run 80000 experiments on a 9 node graph). Each algorithm was given the same graph, distribution model, and object arrangement for each trial.

Figure 96 shows the performance of each algorithm in terms of its average distance traveled across various map sizes. As expected, the results are bounded by the Traveling Salesman and Offline Optimal searches. The Probability-based and Proximity-based heuristics performed almost identically, and were outperformed by the Probability-Proximity algorithm. They are all outperformed by the exhaustive Bayesian search.

The average traveled distance for all the algorithms increases as the map grows in size. Note that the Exhaustive Bayesian search performance curve in Figure 96 is cut off at 8 nodes. For maps larger than 8 nodes, the intractable time complexity of the algorithm precludes timely completion of a full experiment of 80,000 trials, i.e., the time to calculate the path becomes greater than 5 seconds, as seen in Figure 97. There are no statistically significant differences between the mean travel distances of Offline Optimal, Exhaustive Bayesian, and Probability-Proximity searches for environments smaller than 5 nodes, according to a two sample T-Test ($p - value > 0.05$). For environments larger than 5 nodes, there is a statistically significant difference between Offline Optimal, and both Exhaustive Bayesian and Probability-Proximity searches ($p - value < 0.05$). For environments larger that 6 nodes, there is a statistically significant difference between Exhaustive Bayesian and Probability-Proximity searches ($p - value < 0.05$). Figure 97 indicates that the runtime of the heuristic approaches increases linearly with the number of nodes, but the runtime of the Exhaustive Bayesian search increases exponentially (appears linear on the logarithmic scale).

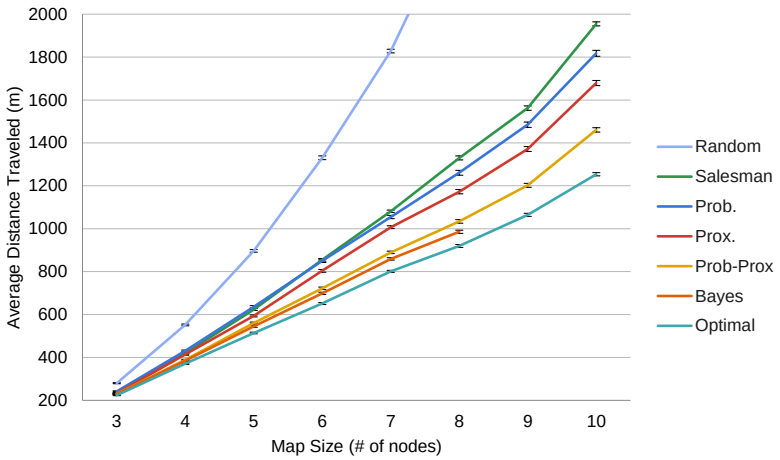


Figure 96: Traveled Distance for Algorithms in Environments with Increasing Complexity.

The Probability-Proximity, Exhaustive Bayesian, and Offline Optimal search algorithms were also tested on a real mobile robot, which is part of the Building-Wide Intelligence project at the University of Texas at Austin [240]. The robot is custom built on a Segway base, with a 2D Hokuyo

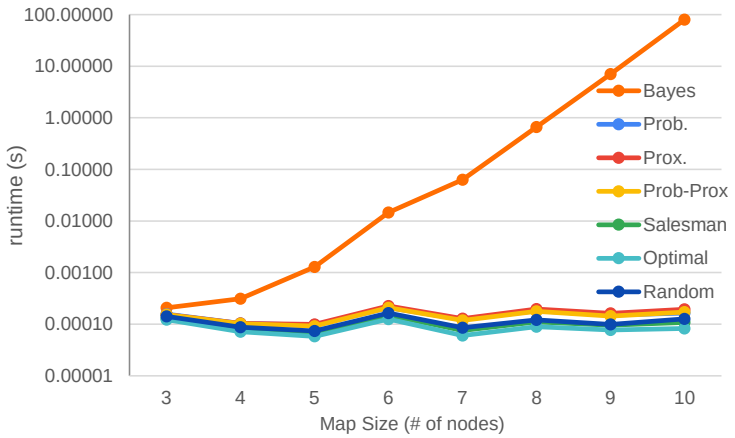


Figure 97: Run-time Comparison of the Algorithms on a Logarithmic Scale.

Lidar used for localization, and a Kinect RGBD camera for perception. The limited computation resources on the robot resulted in long processing times by the computer vision software. To resolve that, we offboarded the object recognition process and had the robot send the feed from its Kinect V1 RGBD camera, to be processed on a separate machine. This workaround enabled the robots to scan as it was moving without having to stop and wait. The STPP provided to the robot, as shown in Figure 98 (a), consisted of 7 location nodes and the following distribution model:

- Object A occurs at node 2 10%, node 3 80%, and node 7 10% of the time.
- Object B occurs at node 1 20%, node 3 50%, and node 7 30% of the time.
- Object C occurs at node 1 20%, node 2 30%, node 4 20% and node 5 30% of the time.
- Object D occurs at node 4 50%, and node 5 50% of the time.

In each experiment, the robot started with the same probability distribution knowledge and location as input to the algorithm. The robot determined the next best location and traveled to it. At the location, it surveyed the area and determined what objects were there. The robot updated the current probabilities to match new knowledge. It repeated these steps, until all objects were found

Ten arrangements of objects were sampled according to the probabilities in the distribution model. The arrangements were replicated in the real world and the three algorithms were tested on each arrangement.

Figure 98 (b) shows the actual path that was generated by each algorithm for an example trial that was conducted on the real robot. In this specific case, Exhaustive Bayesian produced the same path as the Offline Optimal algorithm, which resulted in approximately 60 meters in length. The Probability-Proximity path was much longer with 104.63 meters in length.

Figure 99 summarizes the results of the scavenger hunts that were tested on the robot. Each reported result is the average of ten trials. Like in simulation, Exhaustive Bayesian search outperformed the Probability-Proximity algorithm and was outperformed by Offline Optimal. Due to COVID-19 we were unable to complete the real robot experiments of the DQN algorithm. However, we ran simulated SH experiments with DQN in the simulator we created, on an environment similar to the

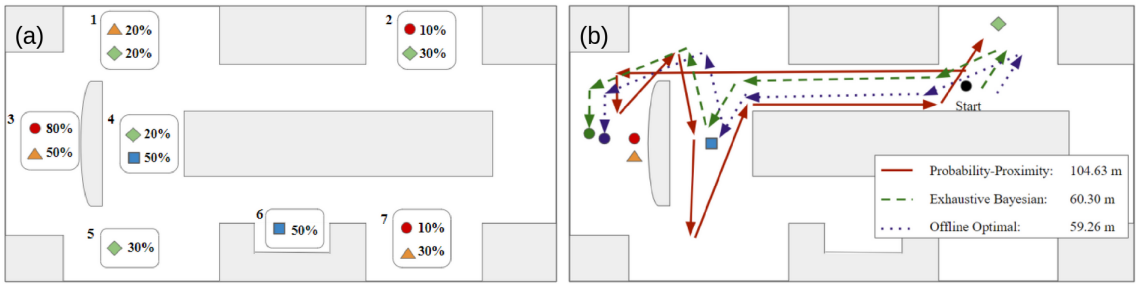


Figure 98: Map of the Lab Space and Example Paths. (a) Map of the lab space used for real-world experiments. The percentages shown are representative of the occurrence model provided to the agent. (b) Examples of the paths generated by each algorithm in a real world experiment.

one where the real robot experiments were run. The edge lengths in the simulated environment were exactly the lengths measured by the real robot's odometer sensor when traversing the same edges in our lab. The results indicate that DQN performs similarly to the Exhaustive Bayesian search algorithm.

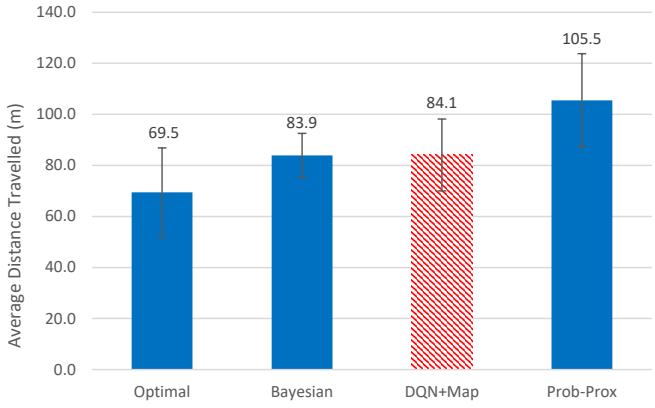


Figure 99: The Average Distance Traveled by Each Algorithm in the Robot Experiments.

4.5.12 Model-based Lifelong Reinforcement Learning with Bayesian Exploration (VBLRL).

Setup We evaluated the performance of VBLRL on HiP-MDP versions of several continuous control tasks from the Mujoco physics simulator [241], *HalfCheetah-gravity, HalfCheetah-bodyparts, Hopper-gravity, Hopper-bodyparts, Walker-gravity, Walker-bodyparts*, all of which are lifelong-RL benchmarks used in prior work [141]. For each of six different domains, the task-specific hidden parameters correspond to different gravity values or different sizes and masses of the simulated body parts.

Compared with prior work, we substantially reduced the number of iterations that the agent can sample and train on: 100 iterations for each task and a horizon of 100 (Halfcheetah) or 400 (Hopper & Walker) for each iteration. We used such settings to increase the difficulty of lifelong learning and

test the sample efficiency of lifelong RL algorithms, given that it is hard for a single-task training algorithm to obtain a good policy within such limited number of interactions with the environments.

Table 38: Results on OpenAI Gym Mujoco Domains

	VBLRL	**T-HiP-MDP**	**LPG-FTW** ($2\times$ **samples**)	**EWC** ($2\times$ **samples**)	**Single-task MBRL**
CG-Start	$\mathbf{160.68 \pm 48.80}$	126.95 ± 31.41	-81.59 ± 9.18	-3426.76 ± 827.99	-83.96 ± 60.10
CG-Train	$\mathbf{226.72 \pm 26.53}$	170.20 ± 39.92	-29.49 ± 11.03	-3440.66 ± 1007.50	-40.47 ± 10.68
CG-Back	$\mathbf{231.79 \pm 23.49}$	97.84 ± 22.04	-29.95 ± 11.64	-6672.33 ± 3748.63	/
CB-Start	$\mathbf{110.74 \pm 41.96}$	78.95 ± 18.43	-263.94 ± 40.80	-5016.93 ± 1708.10	-101.02 ± 39.11
CB-Train	$\mathbf{173.97 \pm 78.26}$	87.20 ± 9.42	-217.86 ± 42.82	-5454.52 ± 2145.82	-58.93 ± 33.24
CB-Back	$\mathbf{181.60 \pm 67.50}$	116.03 ± 17.35	-116.41 ± 65.64	-13889.31 ± 6851.05	/
HG-Start	268.11 ± 33.29	230.21 ± 30.75	305.63 ± 34.55	$\mathbf{306.79 \pm 29.14}$	18.62 ± 1.51
HG-Train	332.89 ± 23.68	285.87 ± 41.48	$\mathbf{352.10 \pm 25.25}$	345.47 ± 40.26	20.46 ± 1.77
HG-Back	$\mathbf{360.94 \pm 7.71}$	312.11 ± 55.45	347.07 ± 44.09	301.06 ± 114.11	/
HB-Start	193.27 ± 8.03	181.94 ± 12.97	$\mathbf{256.13 \pm 69.68}$	133.95 ± 27.61	19.12 ± 1.75
HB-Train	$\mathbf{296.34 \pm 11.15}$	227.50 ± 41.78	285.62 ± 78.18	139.01 ± 46.24	21.25 ± 2.67
HB-Back	$\mathbf{316.74 \pm 20.72}$	213.93 ± 75.95	281.99 ± 74.96	-384.27 ± 199.27	/
WG-Start	$\mathbf{248.97 \pm 19.95}$	217.94 ± 39.99	140.75 ± 67.64	163.61 ± 86.55	4.32 ± 0.23
WG-Train	$\mathbf{333.65 \pm 26.73}$	268.97 ± 36.26	166.00 ± 45.04	181.76 ± 97.42	210.41 ± 39.34
WG-Back	$\mathbf{364.18 \pm 49.60}$	173.33 ± 58.43	168.39 ± 94.19	216.09 ± 62.67	/
WB-Start	$\mathbf{222.88 \pm 34.49}$	220.07 ± 20.31	164.79 ± 10.31	164.00 ± 34.31	4.64 ± 0.23
WB-Train	$\mathbf{311.41 \pm 16.30}$	266.97 ± 26.72	165.75 ± 2.39	206.97 ± 41.40	196.42 ± 22.04
WB-Back	$\mathbf{317.82 \pm 13.90}$	229.10 ± 16.60	195.01 ± 49.25	152.94 ± 70.59	/

We compared VBLRL against state-of-the-art lifelong RL methods LPG-FTW [141], EWC [242], single-task model-based RL baseline (using the same BNN structure and planning procedures) as well as T-HiP-MDP [23]. For a fair comparison, we further replaced the DDQN algorithm [243] used in Killian et al.'s paper with CEM planning, and let the transition model also predict the reward for each state–action pair. This modified baseline is similar to the single-model version of VBLRL (i.e., only using the world-model posterior). Moreover, LPG-FTW and EWC are built upon a relatively simple policy gradient baseline, which does not perform as well as our model-based baseline in the single-task setting within the same amount of interactions. Thus, we let them collect **twice the amount of samples** each iteration compared to VBLRL.

Results The results are shown in Table 38. *CG* denotes **Cheetah-Gravity**, *CB* denotes **Cheetah-Bodyparts**, *HG* denotes **Hopper-Gravity**, *HB* denotes **Hopper-Bodyparts**, *WG* denotes **Walker-Gravity**, *WB* denotes **Walker-Bodyparts** For all six domains, we report the average performance of all the tasks at the beginning of training (**Start**) and after all training for each new task (**Train**), as well as the average performance for all previous tasks after training for a given number of tasks, which is our backward transfer test (**Back**). As shown in the results, our VBLRL shows better performance on all three test stages of the HalfCheetah domain and Walker domain, as well as better backward transfer performance on Hopper-gravity and Hopper-bodyparts than the other three algorithms. Comparing Figure 100 first column and second column, we find that, in the Hopper domains, even though the single-task baseline used by LPG-FTW and EWC performed much better than VBLRL after we let them collect $\mathbf{2}\times$ samples each iteration, VBLRL still achieves comparable performance with LPG-FTW and EWC in lifelong RL experiments, suggesting that VBLRL is capable of making better use of the data in the lifelong learning setting. In the walker domains where the single-task baselines achieved similar performance, VBLRL shows significantly better performance than LPG-FTW/EWC in lifelong RL experiments. These comparisons also suggest that VBLRL's lifelong learning performance may be further improved when combined with better model-based deep RL algorithms, like Dreamer [244].

EWC fails in some of the tasks as it is hard to directly learn a single shared policy that achieves good performance when the tasks are highly diverse. T-HiP-MDP shows good results on some of the tasks because it is more sample-efficient in learning a shared model across all the tasks and more easily captures the world-model uncertainty. However, it cannot achieve as good performance as VBLRL as it is hard to model the task-specific uncertainty using only one model across all tasks, which also leads to negative backward transfer performance on tasks like Cheetah-Gravity. Comparing VBLRL's performance on the **Train** stage and **Back** stage, we also find that it shows positive backward transfer results on most tasks, without showing patterns of catastrophic forgetting. Overall, VBLRL's world-model posterior contributes to better forward transfer performance (**Start**), the learning of task-specific posterior contributes to better forward transfer training for each new task (**Train**), and the combination of these two posteriors guides the agent to achieve better backward transfer performance (**Back**).

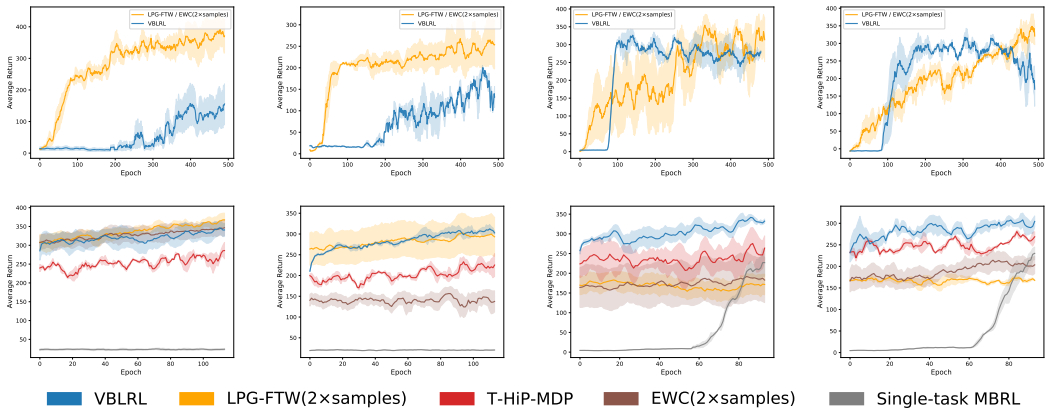


Figure 100: Average Performance Comparison of Single-Task Baselines to Lifelong RL Algorithms. First column: Average performance comparison for single-task baselines. Second Column: Average performance comparison for lifelong RL algorithms. From left to right: Hopper-Gravity; Top-Right: Hopper-Bodyparts; Bottom-Left: Walker-Gravity; Bottom-Right: Walker-Bodyparts.

4.5.13 Possibility Before Utility: Learning and Using Hierarchical Affordances (HAL).

In our experiments we aim to answer the following questions: (1) Does HAL improve learning in tasks with complex dependencies? (2) Is HAL robust to milestone selection and context stochasticity? (3) Can HAL more effectively learn a diverse set of skills when trained task-agnostically?

Environments We evaluate our method, along with several baselines, on two complex environments with intricate subtask dependency structures: CRAFTING and TREASURE. Both environments (visualized in Fig. 101) are extensions of the minigrid framework [170]. Agents receive an egocentric image of the environment, as well as a vector describing their inventory (items picked up from the environment and currently in their possession) as observations. The action spaces are discrete and include actions for turning left/right, and moving forward/backward, as well as environment specific actions detailed below. Task hierarchies, walk-throughs of successful trajectories, and additional information about both environments are included in the original paper's Appendix.

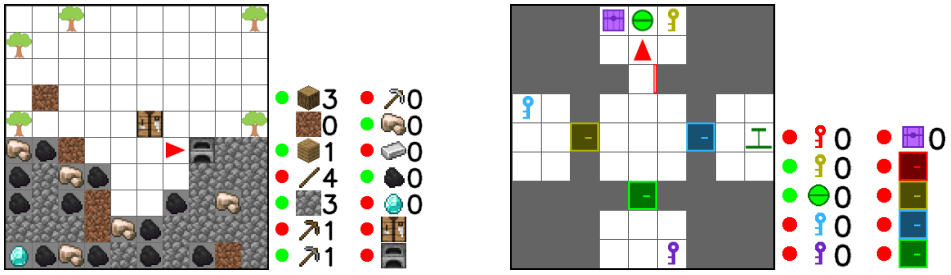


Figure 101: Screenshots of CRAFTING (left) and TREASURE (right) Environments. Displayed to the right of the environments are each item's ground-truth affordance indicator and inventory count.

CRAFTING is based on Minecraft, a popular open-ended video game in which players collect resources from their environment and use them to craft objects which can be used to then obtain more resources. As such, the hierarchy of possible subtasks is immensely complex and presents a significant challenge for AI agents to reach subtasks deeper in the hierarchy. We develop an environment that replicates this hierarchical complexity without the commensurate visuomotor complexity which our method does not aim to address. In addition to movement actions, CRAFTING includes actions to mine the object immediately in front of the agent (which requires an appropriate pickaxe), as well as to craft and smelt the various objects (pickaxes, iron ingots, etc.). The full set of milestones contains items that are either craftable or collectable. CRAFTING naturally contains node stochasticity since the collection of certain items, due to the random procedural generation, requires slightly different milestone trajectories across episodes (e.g. mining variable amount of stone to encounter diamond).

TREASURE is a navigation task that requires the agent to collect various items and use them to unlock rooms to reach further items. The ultimate goal is to unlock a treasure chest, which requires a sequence of collecting several keys, as well as placing an object on a weight-based sensor, in order to open the requisite doors. Agents can only carry one object at a time, so they must reason about which object to pick up based on what it will afford them (e.g. if the weight-based sensor room is locked, the weight object is not currently useful). Like CRAFTING, TREASURE contains node stochasticity due to the procedural generation. For example, the central room that the agent is spawned in can contain either of the red or yellow key individually, or both together. Unlike CRAFTING, which has a large action space to accommodate the various crafting recipes, this environment only contains actions to move and a single "interaction" action that is used to pick up keys, open doors, etc. While CRAFTING has a more complex hierarchy and greater diversity in the potential ordering of subtasks, TREASURE has on average more difficult subtasks. The full set of milestones contains each object the agent can successfully interact with in the environment (e.g. opening door, collecting key).

Baselines Our set of baselines is summarized in Table 39. All methods are based on the Rainbow [245] Deep Q-Learning algorithm, which combines several improvements to vanilla DQNs [166]. To compensate for the lack of milestone signals, non-hierarchical methods use a dense reward function that incorporates milestone signals for the first time each milestone is obtained in the episode. To evaluate the efficacy of our affordance classifier online learning procedure, we compare our method to an "oracle" that differs only by using *ground-truth* affordances for masking subtasks. The node stochasticity inherent to both environments, as well as the edge stochasticity

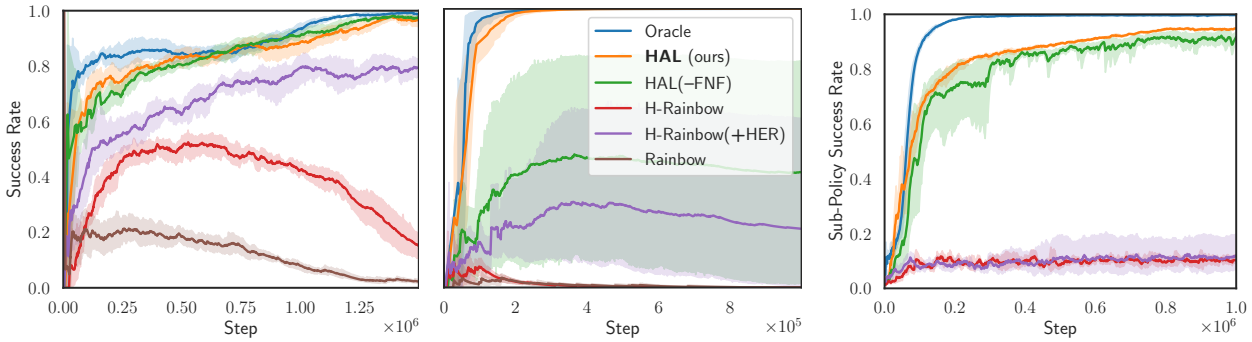


Figure 102: Success Rate over the Course of Training for CRAFTING `iron` Task (left) and TREASURE (center). Sub-policy Success for TREASURE (right). Success rate is the proportion of episode where the agent receives the target milestone, and sub-policy success is how often sub-policies, on average, receive the correct milestone when called, before a timing out or collecting incorrect milestones.

Table 39: Summary of Baselines

	Hier-archical Agent	Afford-ance Mask	Hind-sight Replay	False Negative Filtering
Oracle	✓	*Truth*	✓	N/A
HAL (ours)	✓	Learned	✓	✓
HAL(–FNF)	✓	Learned	✓	×
H-Rainbow	✓	N/A	×	N/A
H-Rainbow (+HER)	✓	N/A	✓	N/A
Rainbow	×	N/A	N/A	N/A

later explored, preclude the use of any methods that reason solely over symbols (i.e. automata-, sketch-, or subtask dependency-based approaches). We also incorporate a version of Hindsight Experience Replay (HER) [246] adapted for the discrete milestone setting, which involves re-using "failed" trajectories that result in the collection of an unintended milestone (given the selected subpolicy) as successful data for the relevant subpolicy. For instance, if an agent accidentally collects iron while executing its wooden pickaxe sub-policy, it can use this trajectory to train its iron sub-policy.

Learning efficacy First, we evaluate the ability of HAL and our baselines to learn successful policies for complex tasks in each environment. Learning curves are shown in Figure 102 (plots depict mean and 95% confidence interval over 5 seeds). In both environments, HAL significantly outperforms the strongest baseline, H-Rainbow(+HER) (HR+H), despite both methods receiving the same information, and performs only slightly worse than the oracle, which has access to ground truth. Incorporating HER into H-Rainbow leads to a significant improvement. False negative filtering (and all other components; see Appendix of original paper) appears crucial for learning in the TREASURE environment, but not as much for CRAFTING, though in both cases filtering improves

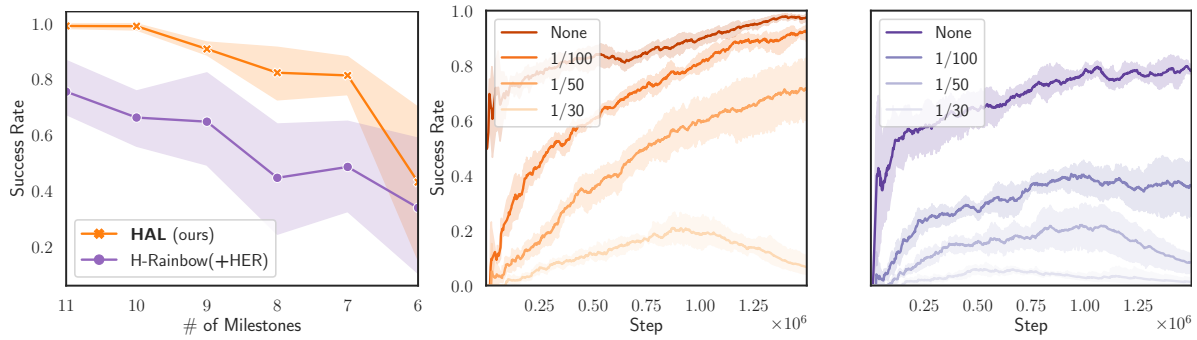


Figure 103: Comparing the Robustness of HAL and HR+H to Varying Milestone Sets (left) and Various Edge Stochasticity Frequencies (center and right, respectively) in CRAFTING `iron` Task.

mask accuracy. Removing false negative filtering causes HAL(-FNF) to be pessimistic, ultimately leading to its unstable learning. Since masking impossible subpolicies would have no impact on an optimal meta-controller, HAL's success must stem from its ability to learn a useful mask *before* TD errors are able to propagate through the meta-controller's Q function. HAL utilizes a more easily learnable function (affordance classifier) to reduce the amount of unnecessary expensive learning (TD error propagation) required. Throughout training, HAL's mask has an impact on greedy subtask selection $\sim$60% of the time, which is evidence that HAL avoids wasting time learning Q-values that the mask is able to prune. Lastly, because affordance-aware methods are more likely to initiate subtasks in an appropriate context, we see they achieve a significantly higher average subpolicy success rate (Fig. 102, right).

Robustness to milestone selection In this section we evaluate HAL's robustness to the selection of milestones. Affordances change when a milestone is removed since that milestone no longer acts as an intermediate link between others. One downside of some approaches that use symbol-based contexts is that an entirely different automata or subtask dependency graph must be defined over the new set of symbols. HAL does not use prior information of this kind, so the learning process is the same across all sets. Figure 103 shows HAL's success on the CRAFTING environment's `iron` task when using "incomplete" milestone sets, relative to the full human-designed set. We see that randomly removing 1 milestone makes no significant difference for HAL, and even after removing 4 milestones, HAL still achieves better performance than HR+H using the full set. HAL's performance drops when 5 milestones are removed likely due to the increased sparsity of the signal (i.e. greater subtask length) and variance in milestone set quality. However, when we double the training time for these same sets, we find that HAL is able to converge to a 97% success rate on at least one set, while HR+H fails to converge on any set and ends with a maximum success rate of around 70%.

Robustness to stochasticity We modify CRAFTING so that at each environment step, there is a certain probability that an item in the inventory will disappear. In order to make the task feasible, rare items are less likely to disappear than common ones. This procedure produces *edge stochasticity*, since the disappearance of items may alter affordances, and this can occur at any time. We test three different levels of stochasticity, and display the learning curves in Figure 103. With a disappearance frequency of $1/100$ that cuts HR+H's success rate in half, HAL is still able to reach

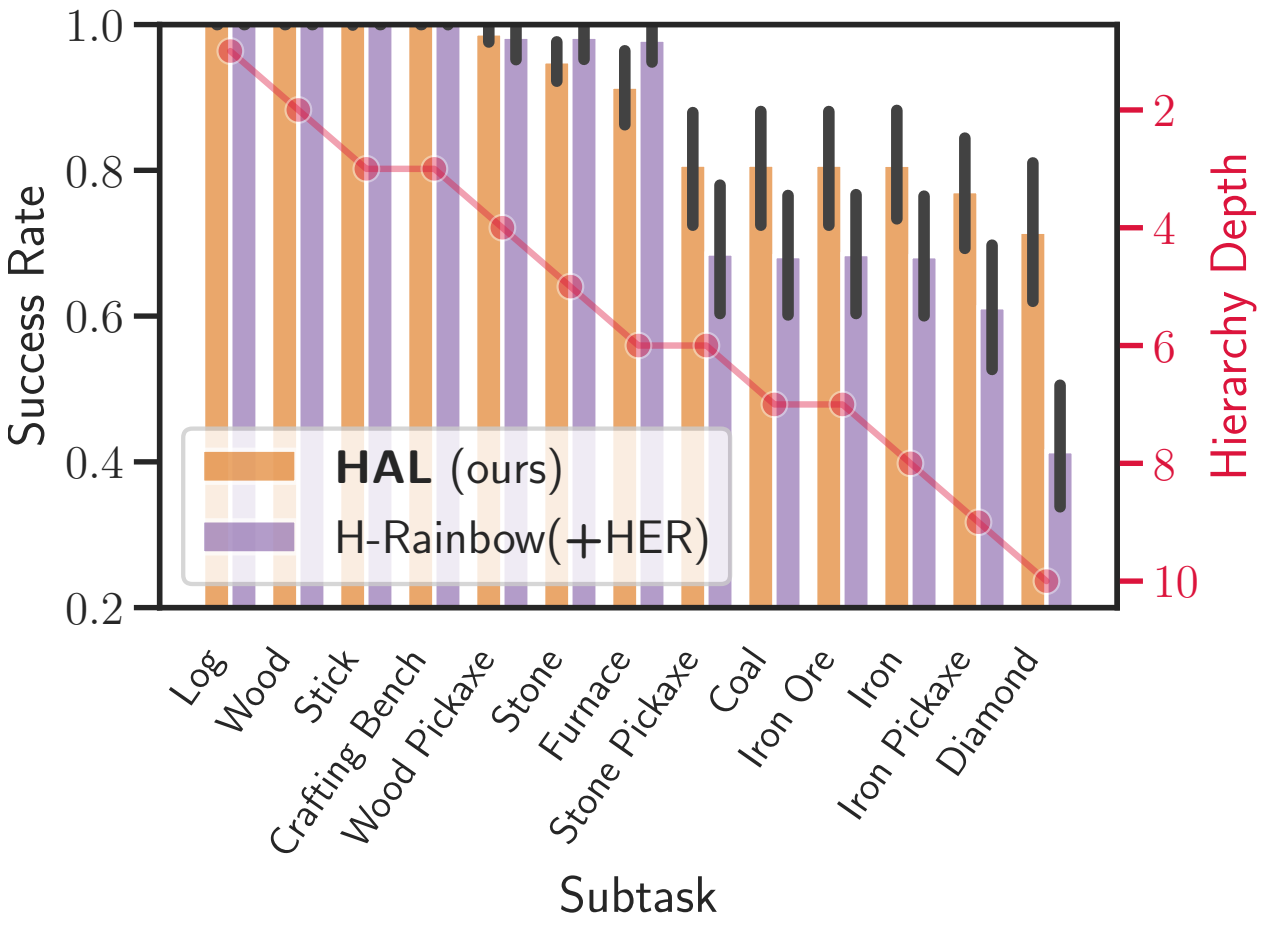


Figure 104: Percentage of Episodes Where each Milestone is Achieved in CRAFTING Environment Task-Agnostic Setting.

its non-stochastic success rate. With a frequency of $1/50$, HAL performs comparably to HR+H with no stochasticity. To put these stochasticity rates into context, the algorithm's average episode length about halfway through training is still over 1000 steps (see Appendix of original paper), meaning *dozens* of items are removed from the agent's inventory over the course of an episode. By learning a model of affordances grounded in the present state, HAL is able to detect and adapt to these stochastic events.

Task-agnostic learning We next test the ability of HAL to learn skills when no task-specific extrinsic rewards (only milestones) are provided by the environment. Since we cannot learn a meta-controller in the absence of rewards, we instead randomly select subtasks with some probability ϵ_{mc}, and random *afforded* subtasks otherwise (only for HAL). We evaluate both HAL and HR+H. In Figure 104 we see that by the end of 10^6 steps, HAL is able to more reliably complete the milestones deeper in the hierarchy in the CRAFTING environment. We note that HR+H is able to marginally outperform HAL in tasks shallower in the hierarchy (e.g. wood pickaxe, stone, furnace), potentially as a result of failing to reach deeper tasks and getting more practice on shallower ones. This result is an indication of the general utility of HAL in environments with complex task hierarchies.

5 CONCLUSIONS

Our L2M project developed numerous fundamental advances for lifelong machine learning, showcasing robust performance across a variety of applications and domains. Our application to service robots demonstrated the capabilities of lifelong learning in the scavenger hunt, and culminated in a live demo of real-time lifelong learning on the service robot to an audience during ICRA 2022. Our project also resulted in methods for visual navigation and exploration that won the 2020 PointNav challenge.

From our project and other projects under the DARPA L2M program, we as a scientific community have a much better understanding of how to enable lifelong machine learning. Concretely, the L2M program spurred major interest and advances in lifelong learning, including cross-pollination of ideas from biology (e.g., the sleep-wake cycle). During the course of the L2M program, publications in major academic machine learning conferences changed from a handful of submissions in 2017 to 100+ submissions to major ML conferences in 2021 (as a concrete number, the first Conference on Lifelong Learning Agents (CoLLAs) in 2022 had 110 submissions; 100 after desk rejections).

This work has also revealed a number of open problems and promising directions for future research related to lifelong learning that may be of interest to DARPA and the DoD; we briefly describe these below:

- **Lifelong learning remains an open research problem, especially in self-supervised and few-shot settings.** Although we now have a broad set of methods for forward/backward transfer and avoiding catastrophic forgetting, and a much better understanding of how to enable lifelong learning in deep networks, there is still a ways to go. As a community, we are still in the infancy of enabling lifelong machine learning algorithms that are robust and truly capable of learning over a lifetime while enabling forward/backward transfer and intelligently maintaining previously learned knowledge and performance on previous tasks. For example, while natural learners (animals and humans) can learn from very little experience or even single episodes, current lifelong learning approaches still rely on extensive experience. It is entirely unrealistic that labels or rewards are provided for such extensive experience, and so instead, a practical lifelong learner needs to operate largely from self-directed or self-supervised mechanisms from mostly unlabeled data. Consequently, as a community, we need to move away from task-incremental supervised settings for lifelong learning and more toward lifelong learning in task-agnostic open worlds. We also need lifelong learning methods that support the more realistic scenarios humans would encounter in those open worlds, such as those having multiple interacting goals, requiring dynamic composition of skills, or having high-level descriptions or instructions.

- **Learning reusable knowledge remains elusive.** Despite our progress in lifelong learning, we do not yet have the ability to acquire truly reusable knowledge. Much of the transfer comes from learning and reusing layered feature representations, yet these are often times overfit to specific tasks. Avoiding forgetting in many cases corresponds to retaining critical pathways for previous tasks and training new tasks along less-used feature pathways. This avoids forgetting while sacrificing potential for forward (and backward) transfer. Instead, we really want to train up robust self-contained representations that are *reusable* across different

tasks, enabling rapid forward transfer and adaptation. For the most part, such ability to learn reusable knowledge remains elusive to current lifelong learning methods. An exception is the use of compositional representations for lifelong learning, as we describe next.

- **Compositional representations show strong promise for knowledge reuse, transfer, and lifelong learning.** Our methods for lifelong learning with compositional representations validated our intuition that, in order for knowledge to be maximally reusable, it should correspond to self-contained units that could be combined in multiple ways with other similar units. We demonstrated in both supervised and reinforcement learning settings that agents capable of decomposing problems into easier subproblems and later combining the solutions to the subproblems are far stronger than standard monolithic learners in terms of zero-shot, forward, and backward transfer. However, we also found in our experiments that these capabilities so far require manual specification of the exact compositional structure of the tasks. Realistic deployments prohibit such specifications, since the exact structure of the tasks' solutions is most often unknown. In consequence, future research should automate the process of discovering compositional structures in a lifelong setting. In addition, work on unifying variations in the definition of compositionality (e.g., functional, temporal, logical, morphological) into integrated approaches that leverage the often complementary benefits of each would lead to highly flexible and adaptive lifelong learners.

- **Transfer between classification and reinforcement learning problems remains a challenge.** We saw in early experiments and in our integrated pipeline (Section 4.3.3) that, despite our intuition that it should work, it is challenging to enable transfer between classification and reinforcement learning tasks. One reason might be the types of features that are learned—classification encourages the acquisition of features that discriminate among classes, while RL encourages action-discriminative features. These are not necessarily compatible, and so complicate transfer across different learning paradigms.

- **Greater human interpretability supports developing lifelong learning agents capable of scaling to higher complexity problems.** In both our individual research and in developing our integrated lifelong learning framework, we often find that certain approaches which are highly effective in toy domains do not scale to high-complexity settings inspired by real-world applications. While we can characterize these successes and failures with measurements of task and learning performance, it can still be difficult to determine precisely why a previously proven approach fails on a new problem. The L2M program's focus on a broader suite of lifelong learning metrics moves the community further towards understanding the behavior of lifelong learners, but a solely metrics-based approach is only an extrinsic indicator of what an agent is actually learning. Research to develop more human-interpretable lifelong learning approaches could advance a more intrinsic understanding of the learned models. Such approaches require advances in representations, such as the compositional representations discussed in this report, through connections to symbolic reasoning, and grounding representations in theories of natural learning advanced by cognitive science. Additionally, putting more emphasis on interpretability supports the development of safer approaches for deploying lifelong learning agents in many real environments.

- **Real-world applications of lifelong learning are needed.** Given its promise, lifelong

learning could benefit numerous real applications. This is especially the case in embodied agents that interact with the world during extended deployments, and so consequently must adapt over time to acquire a variety of skills. Such embodied agents provide an ideal setting for acquiring numerous experience (most of it unsupervised), learning a diverse variety of tasks, and encountering various goals and objectives that much be learned, all while operating in a dynamic and uncertain environment. Such a setting naturally necessitates lifelong learning, and yet also mirrors a variety of useful applications, from autonomous robots, to self-driving vehicles, to intelligent personal assistants.

Lifelong machine learning has great potential, and in our view, is a critical capability necessary for the development of intelligent and robust systems that operate in dynamic environments.

REFERENCES

[1] N. D. Kalka, B. Maze, J. A. Duncan, K. O'Connor, S. Elliott, K. Hebert, J. Bryan, and A. K. Jain, "Ijb–s: Iarpa janus surveillance video benchmark,"

[2] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 304–311, IEEE, 2009.

[3] S. Yang, P. Luo, C. C. Loy, and X. Tang, "WIDER FACE: A face detection benchmark," in *CVPR*, 2016.

[4] S. Lee, J. Stokes, and E. Eaton, "Learning shared knowledge for deep lifelong learning using deconvolutional networks," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2837–2844, 2019.

[5] S. Lee, S. Behpour, and E. Eaton, "Sharing less is more: Lifelong learning in deep networks with selective layer transfer," in *Proceedings of the 38th International Conference on Machine Learning (ICML-21)*, pp. 6065–6075, 2021.

[6] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.

[7] Z. Xu, H. P. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 2396–2407, 2018.

[8] Z. Zheng, J. Oh, and S. Singh, "On learning intrinsic rewards for policy gradient methods," in *Advances in Neural Information Processing Systems*, pp. 4644–4654, 2018.

[9] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," in *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2019.

[10] S. Narvekar and P. Stone, "Generalizing curricula for reinforcement learning," in *4th Lifelong Learning Workshop at the International Conference on Machine Learning (ICML 2020)*, July 2020.

[11] S. Jin, A. RoyChowdhury, H. Jiang, A. Singh, A. Prasad, D. Chakraborty, and E. Learned-Miller, "Unsupervised hard example mining from videos for improved object detection," in *European Conference on Computer Vision (ECCV)*, 2018.

[12] A. RoyChowdhury, P. Chakrabarty, A. Singh, S. Jin, H. Jiang, L. Cao, and E. G. Learned-Miller, "Automatic adaptation of object detectors to new domains using self-training," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 780–790, 2019.

[13] G. Heitz and D. Koller, "Learning spatial context: Using stuff to find things," pp. 30–43, Springer, 2008.

[14] J. Sun and D. W. Jacobs, "Seeing what is not there: Learning context to determine where objects are missing," pp. 1234–1242, IEEE, 2017.

[15] A. Torralba, "Contextual priming for object detection," *International Journal of Computer Vision*, vol. 53, no. 2, pp. 169–191, 2003.

[16] S. Song, A. Zeng, A. X. Chang, M. Savva, S. Savarese, and T. Funkhouser, "Im2pano3d: Extrapolating 360 structure and semantics beyond the field of view," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3847–3856, 2018.

[17] A. Singh, H. Su, S. Jin, H. Jiang, C. Manjesh, G. Luo, Z. Y. He, L. Hong, E. G. Learned-Miller, and R. Cowell, "Half&half: New tasks and benchmarks for studying visual common sense," in *CVPR Workshops*, 2019.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, pp. 6382–6393, 2017.

[19] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence*, 2018.

[20] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 2961–2970, PMLR, 09–15 Jun 2019.

[21] S. Iqbal, C. A. S. De Witt, B. Peng, W. Böhmer, S. Whiteson, and F. Sha, "Randomized entity-wise factorization for multi-agent reinforcement learning," in *International Conference on Machine Learning*, pp. 4596–4606, PMLR, 2021.

[22] F. Doshi-Velez and G. D. Konidaris, "Hidden parameter markov decision processes: A semi-parametric regression approach for discovering latent task parametrizations," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pp. 1432–1440, IJCAI/AAAI Press, 2016.

[23] T. W. Killian, S. Daulton, F. Doshi-Velez, and G. D. Konidaris, "Robust and efficient transfer learning with hidden parameter markov decision processes," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 6250–6261, 2017.

[24] S. Lee, J. Stokes, and E. Eaton, "Learning shared knowledge for deep lifelong learning using deconvolutional networks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 2837–2844, 2019.

[25] J. Mendez, B. Wang, and E. Eaton, "Lifelong policy gradient learning of factored policies for faster training without forgetting," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14398–14409, 2020.

[26] S. Arnold, S. Iqbal, and F. Sha, "When maml can adapt fast and how to assist when it cannot," in *International Conference on Artificial Intelligence and Statistics*, pp. 244–252, PMLR, 2021.

[27] Z. Zheng, J. Oh, M. Hessel, Z. Xu, M. Kroiss, H. Van Hasselt, D. Silver, and S. Singh, "What can learned intrinsic rewards capture?," in *International Conference on Machine Learning*, pp. 11436–11446, PMLR, 2020.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[29] S. K. Ramakrishnan, Z. Al-Halah, and K. Grauman, "Occupancy anticipation for efficient exploration and navigation," in *European Conference on Computer Vision*, pp. 400–418, Springer, 2020.

[30] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, pp. 1–50, 2020.

[31] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[32] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.

[33] A. Kumar and H. Daume III, "Learning task grouping and overlap in multi-task learning," in *Proceedings of the 29th International Conference on Machine Learning*, pp. 1383–1390, Omnipress, July 2012.

[34] A. Maurer, M. Pontil, and B. Romera-Paredes, "Sparse coding for multitask and transfer learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 343–351, June 2013.

[35] P. Ruvolo and E. Eaton, "ELLA: An efficient lifelong learning algorithm," in *Proceedings of the International Conference on Machine Learning (ICML)*, June 2013.

[36] X. Jia, B. D. Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *Advances in Neural Information Processing Systems 29*, pp. 667–675, 2016.

[37] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[38] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *CoRR*, vol. abs/1606.04671, 2016.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[42] H. Bou Ammar, E. Eaton, P. Ruvolo, and M. E. Taylor, "Online multi-task learning for policy gradient methods," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, June 2014.

[43] E. Park and J. B. Oliva, "Meta-Curvature," *arXiv preprint arXiv:1902.03356*, Feb. 2019.

[44] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for Few-Shot learning," *arXiv preprint arXiv:1707.09835*, July 2017.

[45] S. Flennerhag, A. A. Rusu, R. Pascanu, H. Yin, and R. Hadsell, "Meta-Learning with warped gradient descent," *arXiv preprint arXiv:1909.00025*, Aug. 2019.

[46] D. S. Bernstein, *Scalar, Vector, and Matrix Mathematics: Theory, Facts, and Formulas - Revised and Expanded Edition*. Princeton University Press, revised, expanded edition ed., Feb. 2018.

[47] D. S. Chaplot, S. Gupta, D. Gandhi, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural mapping," *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[48] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[49] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[50] K. Fang, A. Toshev, L. Fei-Fei, and S. Savarese, "Scene memory transformer for embodied agents in long-horizon tasks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 538–547, 2019.

[51] S. K. Ramakrishnan, D. Jayaraman, and K. Grauman, "An exploration of embodied visual exploration," *International Journal of Computer Vision*, 2021.

[52] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments," *arXiv preprint arXiv:1712.03931*, 2017.

[53] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[54] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2017.

[55] C. Gan, Y. Zhang, J. Wu, B. Gong, and J. B. Tenenbaum, "Look, listen, and act: Towards audio-visual embodied navigation," *arXiv preprint arXiv:1912.11684*, 2019.

[56] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*, pp. 5–32, Springer, 1992.

[57] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[58] S. Narvekar, J. Sinapov, and P. Stone, "Autonomous task sequencing for customized curriculum design in reinforcement learning," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, August 2017.

[59] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone, "Automatic curriculum graph generation for reinforcement learning agents," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

[60] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, pp. 1312–1320, 2015.

[61] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio, "BabyAI: First steps towards grounded language learning with a human in the loop," in *International Conference on Learning Representations*, 2019.

[62] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, A. V, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, and M. T. Minh, "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," Feb. 2022.

[63] J. Piaget, "Piaget's theory," in *Piaget and His School: A Reader in Developmental Psychology* (B. Inhelder, H. H. Chipman, and C. Zwingmann, eds.), pp. 11–23, Berlin, Heidelberg: Springer, 1976.

[64] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Inferring and executing programs for visual reasoning," in *Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV-17)*, pp. 2989–2998, 2017.

[65] C. Rosenbaum, T. Klinger, and M. Riemer, "Routing networks: Adaptive selection of non-linear functions for multi-task learning," in *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[66] M. Chang, A. Gupta, S. Levine, and T. L. Griffiths, "Automatically composing representation transformations as a means for generalization," in *7th International Conference on Learning Representations (ICLR-19)*, 2019.

[67] V. Pahuja, J. Fu, S. Chandar, and C. Pal, "Structure learning for neural module networks," in *Proceedings of the Beyond Vision and LANguage: inTEgrating Real-world kNowledge (LANTERN)*, pp. 1–10, Association for Computational Linguistics, 2019.

[68] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "PathNet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[69] F. Alet, T. Lozano-Perez, and L. P. Kaelbling, "Modular meta-learning," in *Proceedings of the 2nd Conference on Robot Learning (CoRL-19)*, pp. 856–868, 2018.

[70] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *5th International Conference on Learning Representations (ICLR-17)*, 2017.

[71] L. Kirsch, J. Kunze, and D. Barber, "Modular networks: Learning to decompose neural computation," in *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 2408–2418, 2018.

[72] E. Meyerson and R. Miikkulainen, "Beyond shared hierarchies: Deep multitask learning through soft layer ordering," in *6th International Conference on Learning Representations (ICLR-18)*, 2018.

[73] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 114, no. 13, pp. 3521–3526, 2017.

[74] H. Ritter, A. Botev, and D. Barber, "Online structured Laplace approximations for overcoming catastrophic forgetting," in *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 3738–3748, 2018.

[75] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems 30 (NeurIPS-17)*, pp. 6467–6476, 2017.

[76] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3302–3309, 2018.

[77] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[78] A. N. Gomez, I. Zhang, S. R. Kamalakara, D. Madaan, K. Swersky, Y. Gal, and G. E. Hinton, "Learning sparse networks using targeted dropout," *arXiv preprint arXiv:1905.13678*, 2019.

[79] A. Ozerov, J.-R. Vigouroux, L. Chevallier, and P. Pérez, "On evaluating face tracks in movies," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pp. 3003–3007, IEEE, 2013.

[80] S. Wan, Z. Chen, T. Zhang, B. Zhang, and K.-k. Wong, "Bootstrapping face detection with hard negative examples," *arXiv preprint arXiv:1608.02236*, 2016.

[81] X. Sun, P. Wu, and S. C. Hoi, "Face detection using deep learning: An improved faster rcnn approach," *arXiv preprint arXiv:1701.08289*, 2017.

[82] S. Jin, H. Su, C. Stauffer, and E. Learned-Miller, "End-to-end face detection and cast grouping in movies using erdos-renyi clustering," in *ICCV*, 2017.

[83] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study," *Knowledge and Information Systems*, vol. 42, no. 2, pp. 245–284, 2015.

[84] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *ICML Workshop*, vol. 3, p. 2, 2013.

[85] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4293–4302, 2016.

[86] K.-K. Sung and T. Poggio, "Learning and example selection for object and pattern detection," 1994.

[87] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 761–769, 2016.

[88] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *NIPS*, pp. 91–99, 2015.

[89] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2016.

[90] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[91] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[92] Y. Li, J. Yang, Y. Song, L. Cao, J. Luo, and L.-J. Li, "Learning from noisy labels with distillation.," in *ICCV*, pp. 1928–1936, 2017.

[93] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4068–4076, 2015.

[94] R. C. Gonzalez, R. E. Woods, *et al.*, "Digital image processing," 2002.

[95] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[96] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, p. 4, 2007.

[97] J. Xiao, K. A. Ehinger, A. Oliva, and A. Torralba, "Recognizing scene viewpoint using panoramic place representation," pp. 2695–2702, IEEE, 2012.

[98] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, 2000.

[99] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI-08)*, 2008.

[100] A. Kumar and H. Daumé III, "Learning task grouping and overlap in multi-task learning," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 2012.

[101] H. Bou Ammar, E. Eaton, J. M. Luna, and P. Ruvolo, "Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-15)*, July 2015.

[102] J. Piaget, *The Language and Thought of the Child*. Harcourt, Brace, 1926.

[103] A. Tversky, "Features of similarity," *Psychological review*, vol. 84, no. 4, p. 327, 1977.

[104] C. Luo, J. Zhan, X. Xue, L. Wang, R. Ren, and Q. Yang, "Cosine normalization: Using cosine similarity instead of dot product in neural networks," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN-18)*, pp. 382–391, 2018.

[105] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[106] J. Yoon and S. J. Hwang, "Combined group and exclusive sparsity for deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning (ICML-17)*, pp. 3958–3966, 2017.

[107] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems 29 (NeurIPS-16)*, 2016.

[108] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[109] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," in *Advances in Neural Information Processing Systems 33 (NeurIPS-20)*, pp. 3647–3658, 2020.

[110] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[111] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-12)*, pp. 5026–5033, IEEE, 2012.

[112] J. A. Mendez, H. van Seijen, and E. Eaton, "Modular lifelong reinforcement learning via neural composition," in *10th International Conference on Learning Representations (ICLR-22)*, 2022.

[113] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA-217)*, pp. 2169–2176, 2017.

[114] B. Lake and M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," in *Proceedings of the 35th International Conference on Machine Learning (ICML-18)*, pp. 2873–2882, 2018.

[115] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, pp. 157–163, Elsevier, 1994.

[116] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[117] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 1861–1870, 10–15 Jul 2018.

[118] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[119] J. Oh, V. Chockalingam, H. Lee, *et al.*, "Control of memory, active perception, and action in minecraft," in *International Conference on Machine Learning*, pp. 2790–2799, 2016.

[120] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.

[121] E. Wei, D. Wicke, D. Freelan, and S. Luke, "Multiagent soft q-learning," *arXiv preprint arXiv:1804.09817*, 2018.

[122] F. A. Oliehoek, C. Amato, *et al.*, *A concise introduction to decentralized POMDPs*, vol. 1. Springer, 2016.

[123] C. Schroeder de Witt, J. Foerster, G. Farquhar, P. Torr, W. Boehmer, and S. Whiteson, "Multi-agent common knowledge reinforcement learning," in *Advances in Neural Information Processing Systems 32*, pp. 9927–9939, Curran Associates, Inc., 2019.

[124] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 3744–3753, PMLR, 09–15 Jun 2019.

[125] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposia*, pp. 29–37, 2015.

[126] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[127] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992.

[128] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

[129] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, (Richland, SC), pp. 2085–2087, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[130] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 4295–4304, 10–15 Jul 2018.

[131] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *Proceedings of Machine Learning and Systems (ICML)*, pp. 2611–2622, 2020.

[132] F. A. Oliehoek, M. T. Spaan, N. Vlassis, and S. Whiteson, "Exploiting locality of interaction in factored dec-pomdps," in *Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pp. 517–524, 2008.

[133] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *International Conference on Learning Representations*, 2017.

[134] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, pp. 7254–7264, Curran Associates, Inc., 2018.

[135] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, "Evolutionary population curriculum for scaling multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2020.

[136] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[137] G. Reinelt, *The traveling salesman: computational solutions for TSP applications*, vol. 840. Springer, 2003.

[138] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[139] M. L. Puterman, *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[140] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 6467–6476, 2017.

[141] J. A. M. Mendez, B. Wang, and E. Eaton, "Lifelong policy gradient learning of factored policies for faster training without forgetting," *ArXiv*, vol. abs/2007.07011, 2020.

[142] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993*, pp. 5–13, ACM, 1993.

[143] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, pp. 2348–2356, 2011.

[144] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "VIME: variational information maximizing exploration," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, pp. 1109–1117, 2016.

[145] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *CoRR*, vol. abs/1505.05424, 2015.

[146] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *COLT '93*, 1993.

[147] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped dqn," in *NIPS*, 2016.

[148] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 5331–5340, PMLR, 2019.

[149] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 4759–4770, 2018.

[150] Z. Botev, D. P. Kroese, R. Rubinstein, and P. L'Ecuyer, "Chapter 3 – the cross-entropy method for optimization," *Handbook of Statistics*, vol. 31, pp. 35–59, 2013.

[151] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-16)*, pp. 39–48, 2016.

[152] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf, "Recurrent independent mechanisms," in *9th International Conference on Learning Representations (ICLR-21)*, 2021.

[153] R. Yang, H. Xu, Y. WU, and X. Wang, "Multi-task reinforcement learning with soft modularization," in *Advances in Neural Information Processing Systems 33 (NeurIPS-20)*, pp. 4767–4777, 2020.

[154] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Advances in Neural Information Processing Systems 32 (NeurIPS-19)*, pp. 348–358, 2019.

[155] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proceedings of the 36th International Conference on Machine Learning (ICML-19)*, pp. 2052–2062, 2019.

[156] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, "Benchmarking batch deep reinforcement learning algorithms," *Deep Reinforcement Learning Workshop, NeurIPS-19*, 2019.

[157] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, pp. 181–211, Aug. 1999.

[158] K. Khetarpal, Z. Ahmed, G. Comanici, D. Abel, and D. Precup, "What can I do here? A theory of affordances in reinforcement learning," in *International Conference on Machine Learning*, pp. 5243–5253, PMLR, 2020.

[159] D. Precup, R. S. Sutton, and S. Singh, "Theoretical results on reinforcement learning with temporally abstract options," in *European conference on machine learning*, pp. 382–393, Springer, 1998.

[160] L. Z. Yuan, M. Hasanbeig, A. Abate, and D. Kroening, "Modular deep reinforcement learning with temporal logic specifications," *arXiv preprint arXiv:1909.11591*, 2019.

[161] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *arXiv preprint arXiv:2010.03950*, 2020.

[162] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *International Conference on Machine Learning*, pp. 166–175, PMLR, 2017.

[163] S. Sohn, H. Woo, J. Choi, and H. Lee, "Meta reinforcement learning with autonomous inference of subtask dependencies," in *International Conference on Learning Representations*, 2020.

[164] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, pp. 3675–3683, 2016.

[165] C. J. C. H. Watkins, "Learning from delayed rewards," *PhD thesis, Cambridge University*, 1989.

[166] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[167] H. Song, M. Kim, D. Park, Y. Shin, and J.-G. Lee, "Learning from noisy labels with deep neural networks: A survey," *arXiv preprint arXiv:2007.08199*, 2020.

[168] H. Liu and P. Abbeel, "Behavior from the void: Unsupervised active pre-training," *arXiv preprint arXiv:2103.04551*, 2021.

[169] A. New, M. Baker, E. Nguyen, and G. Vallabha, "Lifelong learning metrics," *arXiv preprint arXiv:2201.08278*, 2022.

[170] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for OpenAI Gym." https://github.com/maximecb/gym-minigrid, 2018.

[171] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[172] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.

[173] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan, "Deep hashing network for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[174] Z. Chen and B. Liu, *Lifelong Machine Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2016.

[175] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *Proceedings of the 10th International Conference on Machine Learning*, pp. 41–48, Morgan Kaufmann, 1993.

[176] J. Yoon, E. Yang, and S. Hwang, "Lifelong learning with dynamically expandable networks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[177] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, and A. A. e. a. Rusu, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[178] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[179] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek, "Benchmark environments for multitask learning in continuous domains," *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*, 2017.

[180] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Proceedings of the 3rd Conference on Robot Learning (CoRL-19)*, 2019.

[181] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, pp. 1332–1338, Dec. 2015.

[182] L. Bertinetto, J. F. Henriques, P. Torr, and A. Vedaldi, "Meta-learning with differentiable closed-form solvers," in *International Conference on Learning Representations*, 2019.

[183] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 3630–3638, Curran Associates, Inc., 2016.

[184] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 1126–1135, PMLR, 2017.

[185] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? towards understanding the effectiveness of {maml}," in *International Conference on Learning Representations*, 2020.

[186] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9068–9079, 2018. Gibson dataset license agreement available at https://storage.googleapis.com/gibson_material/Agreement%20GDS%2006-04-18.pdf.

[187] A. Chang, A. Dai, T. Funkhouser, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," in *Proceedings of the International Conference on 3D Vision (3DV)*, 2017. MatterPort3D dataset license available at: http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf.

[188] Z. Yang, J. Z. Pan, L. Luo, X. Zhou, K. Grauman, and Q. Huang, "Extreme relative pose estimation for rgb-d scans via scene completion," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[189] S. Datta, O. Maksymets, J. Hoffman, S. Lee, D. Batra, and D. Parikh, "Integrating egocentric localization for more realistic pointgoal navigation agents," *CVPR 2020 Embodied AI Workshop*, 2020.

[190] P. Karkus, X. Ma, D. Hsu, L. P. Kaelbling, W. S. Lee, and T. Lozano-Pérez, "Differentiable algorithm networks for composable robot learning," *arXiv preprint arXiv:1905.11602*, 2019.

[191] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," 2020.

[192] "The Habitat Challenge 2020." https://aihabitat.org/challenge/2020/.

[193] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for markov decision processes," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, 2008.

[194] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2778–2787, JMLR. org, 2017.

[195] M. E. Taylor, N. Carboni, A. Fachantidis, I. Vlahavas, and L. Torrey, "Reinforcement learning agents providing advice in complex video games," *Connection Science*, vol. 26, no. 1, pp. 45–63, 2014.

[196] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone, "Source task creation for curriculum learning," in *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, May 2016.

[197] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[198] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[199] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *Proceedings of the 35th International Conference on Machine Learning (ICML-18)*, pp. 4528–4537, 2018.

[200] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European Conference on Computer Vision*, pp. 391–405, Springer, 2014.

[201] L. Zhang, L. Lin, X. Liang, and K. He, "Is faster r-cnn doing well for pedestrian detection?," in *European Conference on Computer Vision*, pp. 443–457, Springer, 2016.

[202] G. Brazil, X. Yin, and X. Liu, "Illuminating pedestrians via simultaneous detection & segmentation," *arXiv preprint arXiv:1706.08564*, 2017.

[203] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv preprint arXiv:1708.02002*, 2017.

[204] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, 2018.

[205] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 4, 2017.

[206] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," *arXiv preprint arXiv:1711.03213*, 2017.

[207] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. Van Gool, "Domain adaptive faster r-cnn for object detection in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3339–3348, 2018.

[208] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," *arXiv preprint arXiv:1409.7495*, 2014.

[209] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron." https://github.com/facebookresearch/detectron, 2018.

[210] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, 2015.

[211] C. Siagian and L. Itti, "Rapid biologically-inspired scene classification using features shared with visual attention," *pami*, vol. 29, pp. 300–312, Jan. 2007.

[212] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," pp. 248–255, Ieee, 2009.

[213] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," 2017.

[214] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," June 2015.

[215] P. Ammirato, P. Poirson, E. Park, J. Kosecka, and A. C. Berg, "A dataset for developing and benchmarking active vision," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[216] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with gaussian processes," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), pp. 19–27, Curran Associates, Inc., 2011.

[217] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st International Conference on Machine Learning (ICML-04)*, 2004.

[218] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *Proceedings of the 9th European Workshop on Reinforcement Learning (EWRL-11)*, 2011.

[219] M. Babes, V. N. Marivate, K. Subramanian, and M. L. Littman, "Apprenticeship learning about multiple intentions," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011.

[220] J. Choi and K. Kim, "Nonparametric Bayesian inverse reinforcement learning for multiple reward functions," in *Advances in Neural Information Processing Systems 25 (NIPS-12)*, 2012.

[221] A. K. Tanwani and A. Billard, "Transfer in inverse reinforcement learning for multiple strategies," in *Proceedings of the 2013 International Conference on Intelligent Robots and Systems (IROS-13)*, IEEE, 2013.

[222] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[223] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[224] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.

[225] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[226] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323–332, 2012.

[227] E. Techapanurak, M. Suganuma, and T. Okatani, "Hyperparameter-free out-of-distribution detection using cosine similarity," in *Proceedings of the Asian Conference on Computer Vision (ACCV-20)*, 2020.

[228] A. Prabhu, P. H. S. Torr, and P. K. Dokania, "GDumb: A simple approach that questions our progress in continual learning," in *Proceedings of the European Conference on Computer Vision (ECCV-20)*, pp. 524–540, 2020.

[229] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, 2018.

[230] J. Achiam, "Spinning up in deep reinforcement learning." https://github.com/openai/spinningup, 2018.

[231] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *AAAI Conference on Artificial Intelligence*, 2018.

[232] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017.

[233] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2186–2188, International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[234] A. Agarwal, S. Kumar, and K. Sycara, "Learning transferable cooperative behavior in multi-agent teams," *arXiv preprint arXiv:1906.01202*, 2019.

[235] T. Wang, H. Dong, V. Lesser, and C. Zhang, "Roma: Multi-agent reinforcement learning with emergent roles," in *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[236] Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, and H. Tang, "Qatten: A general framework for cooperative multiagent reinforcement learning," *arXiv preprint arXiv:2002.03939*, 2020.

[237] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *International Conference on Machine Learning*, pp. 5887–5896, 2019.

[238] S. Hu, F. Zhu, X. Chang, and X. Liang, "{UPD}et: Universal multi-agent {rl} via policy decoupling with transformers," in *International Conference on Learning Representations*, 2021.

[239] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[240] P. Khandelwal, S. Zhang, J. Sinapov, M. Leonetti, J. Thomason, F. Yang, I. Gori, M. Svetlik, P. Khante, V. Lifschitz, *et al.*, "Bwibots: A platform for bridging the gap between ai and human–robot interaction research," *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 635–659, 2017.

[241] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012*, pp. 5026–5033, IEEE, 2012.

[242] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *CoRR*, vol. abs/1612.00796, 2016.

[243] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence,2016*, pp. 2094–2100, AAAI Press, 2016.

[244] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *ArXiv*, vol. abs/1912.01603, 2020.

[245] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[246] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5055–5065, 2017.

APPENDIX A: LIST OF OUR PUBLICATIONS UNDER THE GRANT

Phase 1 Publications

1. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. Iqbal, Shariq, and Fei Sha. ICML 2019.
2. Ad hoc Teamwork with Behavior Switching Agents. M. Ravula, S Alkobi & P. Stone. IJCAI, 2019
3. Building Self-Play Curricula Online by Playing with Expert Agents in Adversarial Games. F. L. Da Silva, A. H. R. Costa & P. Stone. BRACIS 2019.
4. Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning. Iqbal, Shariq, and Fei Sha. ArXiv 2019.
5. Decoupling Adaptation from Modeling with Meta-Optimizers for Meta Learning. Arnold, Sébastien M. R., Shariq Iqbal, and Fei Sha. ArXiv 2019.
6. Deep transfer learning for few-shot SAR image classification. M. Rostami, S. Kolouri, E. Eaton, & K. Kim. Remote Sensing 11 (11), 1374, 2019.
7. Desiderata for Planning Systems in General-Purpose Service Robots. N. Walker, Y. Jiang, M. Cakmak & P. Stone. PlanRob 2019.
8. Discovery of Options via Meta-Gradients. V. Veeriah, T. Zahavy, M. Hessel, Z. Xu, J. Oh, H. Hesselt, D. Silver, & S. Singh. Under Review, 2020.
9. Discovery of Useful Questions as Auxiliary Tasks. V. Veeriah, M. Hessel, Z. Xu, R. Lewis, J. Rajendran, J. Oh, H. Hesselt, D. Silver, & S. Singh. NeurIPS, 2019.
10. Few-Shot Learning via Embedding Adaptation with Set-to-Set Functions. Ye, Han-Jia, Hexiang Hu, De-Chuan Zhan, and Fei Sha. CVPR 2020.
11. Generative Adversarial Imitation from Observation. F. Torabi, G. Warnell & P. Stone. L3-Workshop ICML 2019.
12. How Should an Agent Practice? J. Rajendran, R. Lewis, V. Veeriah, H. Lee, & S. Singh. AAAI, 2019
13. Imitation Learning from Video by Leveraging Proprioception. F. Torabi, G. Warnell & P. Stone. IJCAI 2019.
14. Importance Sampling Policy Evaluation with an Estimated Behavior Policy. J. Hanna, S. Niekum, & P. Stone. ICML 2019.
15. Learning Adaptive Classifiers Synthesis for Generalized Few-Shot Learning. Ye, Han-Jia, Hexiang Hu, De-Chuan Zhan, and Fei Sha. ArXiv 2020.
16. Learning Independently Obtainable Reward Functions. C. Grimm & S. Singh. Arxiv, 2019

17. Learning Shared Knowledge for Deep Lifelong Learning using Deconvolutional Networks. S. Lee, J. Stokes, E. Eaton. IJCAI 2019.

18. Leveraging Human Guidance for Deep Reinforcement Learning Tasks. R. Zhang, F. Torabi, L. Guan, D. H. Ballard & P. Stone. IJCAI 2019.

19. Lifelong Inverse Reinforcement Learning. J. Mendez, S. Shivkumar, E. Eaton. NeurIPS 2018.

20. Multi-Robot Planning with Conflicts and Synergies. Y. Jiang, H. Yedidsion, S. Zhang, G. Sharon & P. Stone. Autonomous Robots, 2019.

21. Open-World Reasoning for Service Robots. Y. Jiang, N. Walker, J. Hart & P. Stone. ICAPS 2019.

22. Optimal Use of Verbal Instructions for Multi-robot Human Navigation Guidance. H. Yedidsion, J. Deans, S. Connor, M. Chillara, J. Hart, Justin, P. Stone & R. J. Mooney. ICSR 2019

23. Recent Advances in Imitation Learning from Observation. F. Torabi, G. Warnell & P. Stone. IJCAI 2019.

24. RIDM: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration. B. S. Pavse, F. Torabi, J.h Hanna, G. Warnell & P.Stone. L3- Workshop ICML 2019.

25. Sample-efficient Adversarial Imitation Learning from Observation. F. Torabi, S. Geiger, G. Warnell & P. Stone. L3- Workshop ICML 2019.

26. SAR image classification using few-shot cross-domain transfer learning. M. Rostami, S. Kolouri, E. Eaton, & K. Kim. CVPR, 2019.

27. Solving Service Robot Tasks: UT Austin Villa@Home 2019 Team Report. R. Shah, Y. Jiang, H. Karnan, G. Briscoe-Martinez, D. Mulder, R. Gupta, R. Schlossman, M. Murphy, J. Hart, L. Sentis & P. Stone. AI-HRI Workshop AAAI Fall Symposium 2019

28. Synthesized Policies for Transfer and Adaptation across Tasks and Environments. Hu, Hexiang, Liyu Chen, Boqing Gong, and Fei Sha. NeurIPS 2018.

29. Task-Motion Planning with Reinforcement Learning for Adaptable Mobile Service Robots. Y. Jiang, F. Yang, S. Zhang & P. Stone. IROS 2019.

30. Teaching a robot tasks of arbitrary complexity via human feedback. G. Wang, C. Trimbach, J.K. Lee, M. Ho, & M.L. Littman. Human Robot Interaction 2020.

31. Transfer Learning via Minimizing the Performance Gap Between Domains. J. Mendez, B. Wang, M. B. Cai, E. Eaton. NeurIPS 2019.

32. Using Task Descriptions in Lifelong Machine Learning for Improved Performance and Zero-Shot Transfer. M. Rostami, D. Isele, & E. Eaton. Journal of AI Research 67, 2020.

33. What Can Intrinsic Rewards Capture? Z. Zhang, J. Oh, M. Hessel, Z. Xu, M. Kroiss, H. Hasselt, D. Silver & S. Singh. Under Review, 2020

34. Zero-Shot Image Classification Using Coupled Dictionary Embedding. M. Rostami, S. Kolouri, Z. Murez, Y. Owekcho, E. Eaton, & K. Kim. arXiv preprint 1906.10509, 2019.

Phase 2 Publications

1. A Lifelong Learning Approach to Mobile Robot Navigation. Bo Liu, Xuesu Xiao, and Peter Stone. IEEE Robotics and Automation Letters (RA-L), 6(2), April 2021. Presented at IEEE International Conference on Robotics and Automation (ICRA)

2. A Penny for Your Thoughts: The Value of Communication in Ad Hoc Teamwork. Reuth Mirsky, William Macke, Andy Wang, Harel Yedidsion, and Peter Stone. In Proceedings of the 29th International Joint Conference on Artificial Intelligence, July 2020.

3. A Scavenger Hunt for Service Robots. Harel Yedidsion, Jennifer Suriadinata, Zifan Xu, Stefan Debruyn, and Peter Stone. In Proceedings of the 2021 International Conference on Robotics and Automation (ICRA 2021), May 2021.

4. Adversarial Intrinsic Motivation for Reinforcement Learning. Ishan Durugkar, Mauricio Tec, Scott Niekum, and Peter Stone. In Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS 2021), December 2021.

5. Agent-Based Markov Modeling for Improved COVID-19 Mitigation Policies. Roberto Capobianco, Varun Kompella, James Ault, Guni Sharon, Stacy Jong, Spencer Fox, Lauren Meyers, Peter R. Wurman, and Peter Stone. The Journal of Artificial Intelligence Research (JAIR), 71:953–92, August 2021.

6. An Exploration of Embodied Visual Exploration. S. Ramakrishnan, D. Jayaraman, K. Grauman. IJCV 2021.

7. An Imitation from Observation Approach to Transfer Learning with Dynamics Mismatch. Siddarth Desai, Ishan Durugkar, Haresh Karnan, Garrett Warnell, Josiah Hanna, and Peter Stone. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS 2020), December 2020.

8. APPLD: Adaptive Planner Parameter Learning from Demonstration. Xuesu Xiao, Bo Liu, Garrett Warnell, Jonathan Fink, and Peter Stone. IEEE Robotics and Automation Letters (RA-L), June 2020.

9. APPLE: Adaptive Planner Parameter Learning From Evaluative Feedback. Zizhao Wang, Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. IEEE Robotics and Automation Letters (RA-L), October 2021.

10. APPLI: Adaptive Planner Parameter Learning From Interventions. Zizhao Wang, Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. In Proceedings of the International Conference on Robotics and Automation (ICRA 2021), May 2021.

11. APPLR: Adaptive Planner Parameter Learning from Reinforcement. Zifan Xu, Gauraang Dhamankar, Anirudh Nair, Xuesu Xiao, Garrett Warnell, Bo Liu, Zizhao Wang, and Peter Stone. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA 2021), June 2021.

12. Balancing Individual Preferences and Shared Objectives in Multiagent Reinforcement Learning. Ishan Durugkar, Elad Liebman, and Peter Stone. In Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), July 2020.

13. Bayesian Exploration for Lifelong Reinforcement Learning: NeurIPS 2021 Deep Reinforcement Learning workshop.

14. Capturing Skill State in Curriculum Learning for Human Skill Acquisition. Keya Ghonasgi, Reuth Mirsky, Sanmit Narvekar, Bharath Masetty, Adrian M. Haith, Peter Stone, and Ashish D. Deshpande. In International Conference on Intelligent Robots and Systems (IROS), September 2021.

15. Coach-Player Multi-Agent Reinforcement Learning for Dynamic Team Composition. Bo Liu, Qiang Liu, Peter Stone, Animesh Garg, Yuke Zhu, and Animashree Anandkumar. In Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021 (ICML), July 2021.

16. CompoSuite: A Compositional Reinforcement Learning Benchmark. Jorge A Mendez, Marcel Hussing, Meghna Gummadi, Eric Eaton. CoLLAs, 2022.

17. Conflict-Averse Gradient Descent for Multi-task learning. Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. In Conference on Neural Information Processing Systems (NeurIPS), 2021, December 2021.

18. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Journal of Machine Learning Research, 21(181):1–50, 2020.

19. DEALIO: Data-Efficient Adversarial Learning for Imitation from Observation. Faraz Torabi, Garrett Warnell, and Peter Stone. In Proceedings of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), September 2021.

20. Deep R-Learning for Continual Area Sweeping. Rishi Shah, Yuqian Jiang, Justin Hart, and Peter Stone. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020), October 2020.

21. Discovery of Options via Meta-Learned Subgoals. Veeriah, Vivek, et al. arXiv preprint arXiv:2102.06741 (2021) (accepted at Neurips 2021)

22. Embedding Adaptation is Still Needed for Few-Shot Learning. Arnold, S.M.R. and Sha, F. In submission, 2021.

23. Environment Predictive Coding for Visual Navigation. S. Ramakrishnan, T. Nagarajan, Z. Al-Halah, K. Grauman. ICLR, 2022.

24. Expected Value of Communication for Planning in Ad Hoc Teamwork. William Macke, Reuth Mirsky, and Peter Stone. In Proceedings of the 35th Conference on Artificial Intelligence (AAAI), February 2021.

25. Firefly Neural Architecture Descent: a General Approach for Growing Neural Networks. Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. In Advances in Neural Information Processing Systems 34 (2020), December 2020.

26. From Agile Ground to Aerial Navigation: Learning from Learned Hallucination. Zizhao Wang, Xuesu Xiao, Alexander J Nettekoven, Kadhiravan Umasankar, Anika Singh, Sriram Bommakanti, Ufuk Topcu, and Peter Stone. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2021), October 2021.

27. Generalizing Curricula for Reinforcement Learning. Sanmit Narvekar and Peter Stone. In 4th Lifelong Learning Workshop at the International Conference on Machine Learning (ICML 2020), July 2020.

28. Goal Blending for Responsive Shared Autonomy in a Navigating Vehicle. Yu-Sian Jiang, Garrett Warnell, and Peter Stone. In Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI), Feb 2021.

29. Grounded Action Transformation for Sim-to-Real Reinforcement Learning. Josiah P. Hanna, Siddharth Desai, Haresh Karnan, Garrett Warnell, and Peter Stone. Special Issue on Reinforcement Learning for Real Life, Machine Learning, 2021, May 2021.

30. Importance Sampling in Reinforcement Learning with an Estimated Behavior Policy. Josiah P. Hanna, Scott Niekum, and Peter Stone. Machine Learning (MLJ), 110:1267–1317, May 2021.

31. Incorporating Gaze into Social Navigation. Justin Hart, Reuth Mirsky, Xuesu Xiao, and Peter Stone. In Robotics: Science and Systems Workshop on Social Robot Navigation (RSS), July 2021.

32. Is the Cerebellum a Model-Based Reinforcement Learning Agent?. Bharath Masetty, Reuth Mirsky, Ashish D. Deshpande, Michael Mauk, and Peter Stone. In Adaptive and Learning Agents Workshop at AAMAS, May 2021.

33. Learning Affordance Landscapes for Interaction Exploration in 3D Environments. T. Nagarajan and. K. Grauman. NeurIPS 2020 (Spotlight)

34. Learning and Reasoning for Robot Dialog and Navigation Tasks. Keting Lu, Shiqi Zhang, Peter Stone, and Xiaoping Chen. In Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pp. 107–117, Association for Computational Linguistics, 1st virtual meeting, July 2020.

35. Learning State Representations from Random Deep Action-conditional Predictions. Zheng, Zeyu, et al. arXiv preprint arXiv:2102.04897 (2021) (accepted at NeurIPS 2021)

36. Learning to Generalize Compositionally by Transferring Across Semantic Parsing Tasks. Zhu, W., Shaw, P., Linzen, T., and Sha, F. In submission, 2021.

37. Learning to Improve Multi-Robot Hallway Navigation. Jin-Soo Park, Brian Tsang, Harel Yedidsion, Garrett Warnell, Daehyun Kyoung, and Peter Stone. In Proceedings of the 4th Conference on Robot Learning (CoRL), November 2020.

38. Learning to Set Waypoints for Audio-Visual Navigation. C. Chen, S. Majumder, Z. Al-Halah, R. Gao, S. Ramakrishnan, K. Grauman. ICLR 2021.

39. Lifelong learning of compositional structures. Jorge Mendez and Eric Eaton. ICLR 2021.

40. Lifelong policy gradient learning of factored policies for faster training without forgetting. Jorge Mendez, Boyu Wang, and Eric Eaton. NeurIPS 2020.

41. Lucid Dreaming for Experience Replay: Refreshing Past States with the Current Policy. Yunshu Du, Garrett Warnell, Assefaw Gebremedhin, Peter Stone, and Matthew E. Taylor. Neural Computing and Applications, May 2021.

42. Machine Learning Methods for Local Motion Planning: A Study of End-to-End vs. Parameter Learning. Zifan Xu, Xuesu Xiao, Garrett Warnell, Anirudh Nair, and Peter Stone. In Proceedings of the 2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2021), October 2021.

43. Machine versus Human Attention in Deep Reinforcement Learning Tasks. Sihang Guo, Ruohan Zhang, Bo Liu, Yifeng Zhu, Mary Hayhoe, Dana Ballard, and Peter Stone. In Conference on Neural Information Processing Systems (NeurIPS, 2021), December 2021.

44. Mechanism Design for Correlated Valuations: Efficient Methods for Revenue Maximization. Michael Albert, Vincent Conitzer, Giuseppe Lopomo, and Peter Stone. Operations Research, March 2021.

45. Modular Lifelong Reinforcement Learning via Neural Composition. Jorge Mendez, Harm van Seijen, and Eric Eaton. ICLR 2022.

46. Motion Planning and Control for Mobile Robot Navigation Using Machine Learning: a Survey. Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Autonomous Robots, February 2022.

47. Move2Hear: Active Audio-Visual Source Separation. S. Majumder, Z. Al-Halah, K. Grauman. ICCV 2021.

48. Occupancy Anticipation for Efficient Exploration and Navigation. S. Ramakrishnan, Z. Al-Halah, K. Grauman. ECCV 2020 (Spotlight). [winner of the 2020 Habitat PointNav Challenge]

49. On Sampling Error in Batch Action-Value Prediction Algorithms. Brahma S. Pavse, Josiah P. Hanna, Ishan Durugkar, and Peter Stone. In the Offline Reinforcement Learning Workshop at Neural Information Processing Systems (NeurIPS), December 2020., December 2020.

50. Pairwise Weights for Temporal Credit Assignment. Zheng, Zeyu, et al. arXiv preprint arXiv:2102.04999 (2021) (in submission to AAAI 2022)

51. PONI: Potential Functions for ObjectGoal Navigation with Interaction-free Learning. S. Ramakrishnan, D. S. Chaplot, Z. Al-Halah, J. Malik, K. Grauman. CVPR, 2022

52. Possibility Before Utility: Learning and Using Hierarchical Affordances. Costales, R., Iqbal, S., and Sha, F. ICLR, 2022.

53. RAIL: A modular framework for Reinforcement-learning-based Adversarial Imitation Learning. Eddy Hudson, Garrett Warnell, and Peter Stone. In Autonomous Robots and Multirobot Systems Workshop at the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), May 2021.

54. Reasoning about Human Behavior in Ad Hoc Teamwork. Jennifer Suriadinata, William Macke, Reuth Mirsky, and Peter Stone. In Adaptive and learning Agents Workshop at AAMAS 2021, May 2021.

55. Recent Advances in Leveraging Human Guidance for Sequential Decision-Making Tasks. Ruohan Zhang, Faraz Torabi, Garrett Warnell, and Peter Stone. Autonomous Agents and Multi-Agent Systems, 35(31), June 2021.

56. Reducing Sampling Error in Batch Temporal Difference Learning. Brahma Pavse, Ishan Durugkar, Josiah Hanna, and Peter Stone. In Proceedings of the 37th International Conference on Machine Learning (ICML), July 2020.

57. Reinforced Grounded Action Transformation for Sim-to-Real Transfer. Haresh Karnan, Siddharth Desai, Josiah P. Hanna, Garrett Warnell, and Peter Stone. In IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 2020), October 2020.

58. Reinforcement Learning of Implicit and Explicit Control Flow in Instructions. Brooks, Ethan A., et al. arXiv preprint arXiv:2102.13195 (2021) (accepted at ICML 2021).

59. Reward (Mis)design for Autonomous Driving. W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. arXiv preprint arXiv:2104.13906, 2021.

60. RIDM: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration. Brahma Pavse, Faraz Torabi, Josiah Hanna, Garrett Warnell, and Peter Stone. IEEE Robotics and Automation Letters (RA-L), 5:6262–69, October 2020.

61. Scalable Multiagent Driving Policies For Reducing Traffic Congestion. Jiaxun Cui, William Macke, Harel Yedidsion, Aastha Goyal, Daniel Urieli, and Peter Stone. In Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), May 2021.

62. Self-supervised learning of optical flow for motion understanding (In preparation)

63. Semantic Audio-Visual Navigation. C. Chen, Z. Al-Halah, K. Grauman. CVPR 2021.

64. Sequential Online Chore Division for Autonomous Vehicle Convoy Formation. Harel Yedidsion, Shani Alkoby, and Peter Stone. Technical Report arXiv e-Prints 2104.04159, arXiv, 2021.

65. Shaping Embodied Agent Behavior with Activity-context Priors from Egocentric Video. T. Nagarajan, K. Grauman. NeurIPS 2021 (Spotlight)

66. Sharing Less is More: Lifelong Learning in Deep Networks with Selective Layer Transfer. Seungwon Lee, Sima Behpour, and Eric Eaton. ICML 2021

67. SHELS: Exclusive Feature Sets for Novelty Detection and Continual Learning Without Class Boundaries. Meghna Gummadi, David Kent, Jorge A Mendez, Eric Eaton. CoLLAs, 2022

68. SoundSpaces: Audio-Visual Navigation in 3D Environments. C. Chen, U. Jain, C. Schissler, S. V. A. Gari, Z. Al-Halah, V. Ithapu, P. W Robinson, K. Grauman. ECCV 2020 (Spotlight)

69. Stochastic Grounded Action Transformation for Robot Learning in Simulation. Siddharth Desai, Haresh Karnan, Josiah P. Hanna, Garrett Warnell, and Peter Stone. In IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 2020), October 2020.

70. Team Orienteering Coverage Planning with Uncertain Reward. Bo Liu, Xuesu Xiao, and Peter Stone. In International Conference on Intelligent Robots and Systems (IROS), 2021, September 2021.

71. Temporal-Logic-Based Reward Shaping for Continuing Reinforcement Learning Tasks. Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. In Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021), February 2021.

72. The PETLON Algorithm to Plan Efficiently for Task-Level-Optimal Navigation. Shih-Yun Lo, Shiqi Zhang, and Peter Stone. The Journal of Artificial Intelligence Research (JAIR), 67, October 2020. Contains material that was previously published in an AAMAS-18 paper (awarded the Best Robotics Paper Award at AAMAS 2018)

73. The Seeing-Eye Robot Grand Challenge: Rethinking Automated Care. Reuth Mirsky and Peter Stone. In Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), May 2021.

74. Using Human-Inspired Signals to Disambiguate Navigational Intentions. Justin Hart, Reuth Mirsky, Xuesu Xiao, Stone Tejeda, Bonny Mahajan, Jamin Goo, Kathryn Baldauf, Sydney Owen, and Peter Stone. In Proceedings of the 12th International Conference on Social Robotics (ICSR), November 2020.

75. VisualEchoes: Spatial Image Representation Learning through Echolocation. R. Gao, C. Chen, Z. Al-Halah, C. Schissler, K. Grauman. ECCV 2020.

76. Visually Grounded Concept Composition. Zhang, B., Hu, H., Qiu, L., Shaw, P., and Sha, F. Findings of EMNLP, 2021.

77. Watch Where You're Going! Gaze and Head Orientation as Predictors for Social Robot Navigation. Blake Holman, Abrar Anwar, Akash Singh, Mauricio Tec, Justin Hart, and Peter Stone. In Proceedings of the International Conference on Robotics and Automation (ICRA 202), May 2021.

78. When MAML can adapt fast and how to assist when it cannot. Arnold, S, Iqbal, S, Sha, F. AISTATS 2021

79. Zero Experience Required: Plug & Play Modular Transfer Learning for Semantic Visual Navigation. Z. Al-Halah, S. Ramakrishnan, K. Grauman. CVPR, 2022.

APPENDIX B: IMPLEMENTATIONS OF STAND-ALONE ALGORITHMS AND COMPONENTS

In addition to our integrated L2M systmem, our project produced a number of stand-alone algorithms and components, as described in Section 3 of the main report. Links to the source code for the stand-alone components is included below along with their estimated technology readiness level (TRL).

ELEMENT	TRL	REPOSITORY
Integrated Algorithms		
Integrated L2M System	4	https://github.com/Lifelong-ML/DARPA-L2M
DF-CNN	5	https://github.com/Lifelong-ML/DF-CNN
DF-CNN/LASEM	5	https://github.com/Lifelong-ML/LASEM
LPG-FTW	5	https://github.com/Lifelong-ML/LPG-FTW
KFO	4	https://github.com/Sha-Lab/kfo
OCCANT	5	https://github.com/facebookresearch/OccupancyAnticipation
LIRPG	4	https://github.com/Hwhitetooth/lirpg
Scavenger hunt API	4	https://github.com/utexas-bwi/scavenger_hunt_api
Other Methods		
COMPCLF	4	https://github.com/Lifelong-ML/Mendez2020Compositional
COMPRL	4	https://github.com/Lifelong-ML/Mendez2022ModularLifelongRL
DETFLICK	4	http://vis-www.cs.umass.edu/unsupVideo/
STSL	4	http://vis-www.cs.umass.edu/unsupVideo/
ELIRL	4	https://github.com/Lifelong-ML/ELIRL
SHELS	4	https://github.com/Lifelong-ML/SHELS
CompoSuite	5	https://github.com/Lifelong-ML/CompoSuite
MAAC	4	https://github.com/shariqiqbal2810/MAAC
REFIL	4	https://github.com/shariqiqbal2810/REFIL
FEAT	4	https://github.com/Sha-Lab/FEAT
Embodied Exploration	5	https://github.com/facebookresearch/exploring_exploration
SoundSpaces	5	https://github.com/facebookresearch/sound-spaces/

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

A2C	Advantage Actor Critic
AI	Artificial Intelligence
ANIL	Almost No Inner Loop
ANS	Active Neural SLAM
AuROC	Area Under the Receiver Operating Characteristic
BT	Backward Transfer
CIFAR	Canadian Institute for Advanced Research
CMAC	Cerebellar Model Arithmetic Computer
CMDP	Curriculum Markov Decision
Process CNN	Convolutional Neural Network
CyCADA	Cycle-Consistent Adversarial Domain Adaptation
DARPA	Defense Advanced Research Projects Agency
DD-PPO	Decentralized Distributed Proximal Policy Optimization
DF-CNN	Deconvolutional Factorized CNN
DoD	Department of Defense
DQN	Deep Q-Network
ELIRL	Efficient Lifelong Inverse Reinforcement Learning
FoV	Field of View
FT	Forward Transfer
GPIRL	Gaussian Process Inverse Reinforcement Learning
GPU	Graphics Processing Unit
GTSRB	German Traffic Sign Recognition Benchmark
HiP-MDP	Hidden-parameter Markov Decision Process
ICRA	International Conference on Robotics and Automation
ID	In-Distribution
IRL	Inverse Reinforcement Learning

LFD	Learning from Demonstration
LPG-FTW	Lifelong policy gradients: faster training without forgetting
MAML	Model-Agnostic Meta-Learning
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MS-COCO	Microsoft Common Objects in Context
MTL	Multi-Task Learning
ODIN	Out-of-Distribution Detector for Neural Networks
OHEM	Online Hard Example Mining
OOD	Out-of-Distribution
PG	Policy Gradient
PM	Performance Maintenance
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RGB-D	Red, Green, Blue, Depth
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROC	Receiver Operator Characteristic
ROS	Robot Operating System
RP	Relative Performance
SE	Sample Efficiency
SH	Scavenger Hunt
SHELS	Sparse High-level Exclusive Low-level Shared
SLAM	Simultaneous Localization and Mapping
STE	Single-Task Expert
STL	Single-Task Learning
SVHN	Street View House Numbers

WIDER	Web Image Dataset for Event Recognition
YOLO	You only look once