DRAPER®

Final Technical Report

Program #: HR0011-19-S-0030
Program Title: Competency-Aware Machine Learning (CAML)
Performer Contract #: HR0011-20-C-0032
Performer Project Title: ALPACA: Autonomous Learning with Probability & Abstraction for Competency Awareness
Performance Period: October 2019 – January 2023
Submission Date: 30-Jan-2023
Draper Reference #: CON06418

Submitted by:

The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139-3563

Submitted to:

(a) DARPA/Contracts Management Office (CMO)
Attn: Catherine Stevens, Contracting Officer
Email: ReportsDSO@darpa.mil

(b) DARPA/Defense Sciences Office
Attn: Dr. Lael Rudd, Program Manager
Email: lael.rudd@darpa.mil

(c) DARPA/DSO
Attn: ADPM
Email: DSO_ADPM@darpa.mil

(d) Contacting Officer's Representative (COR) / AFRL
Attn: Benjamin Lewis
Email: benjamin.lewis.13@us.af.mil

(e) DARPA/Research Support Center
Email: Researchsupport@darpa.mil

(f) DARPA/CMO Closeout
Email: CMO_closeout@darpa.mil

(g) DARPA VAULT
https://vault.darpa.mil

Technical Point of Contact
Dr. Rebecca Russell
Phone: (617) 258-1582
Email: rrussell@draper.com

Programmatic Point of Contact
Mr. Michael R. Crystal
Phone: (617) 258-1039
Email: mcrystal@draper.com

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
30-Jan-2023	Final Technical Report	11-Oct-2019	30-Jan-2023

4. TITLE AND SUBTITLE

Final Technical Report: Autonomous Learning with Probability & Abstraction for Competency Awareness (ALPACA)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
HR0011-20-C-0032	N/A	N/A
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	N/A

6. AUTHOR(S)

Dr. Rebecca L. Russell, Draper, et. al.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
The Charles Stark Draper Laboratory, Inc. (Draper) 555 Technology Sq. Cambridge, MA 02139-3539	CON06418-FTR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
DARPA/CMO 675 N. Randolph Street Arlington VA 22203-2114	DARPA/DSO	N/A

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

None.

14. ABSTRACT

This report describes research carried out by the Draper team as part of the DARPA Competency-Aware Machine Learning (CAML) program. Draper teamed with subcontractors UT Austin, ASU, and CU Boulder to develop ALPACA (Autonomous Learning with Probability & Abstraction for Competency Awareness), a general framework for competency-aware autonomous agents, particularly those based on reinforcement learning (RL). ALPACA provides insight into RL agent competencies and empowers users to examine and constrain agent behavior, facilitating trust building with human teammates and dramatically improving safety for real-world applications.

15. SUBJECT TERMS

Machine Learning, ML, Artificial Intelligence, AI, Explainability, Competency, Human Machine Interaction, HMI

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

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Michael R. Crystal	(617) 258-1039

Table of Contents

1 Summary

This report describes research carried out by the Draper team as part of the DARPA Competency-Aware Machine Learning (CAML) program under contract HR0011-20-C-0032. Draper teamed with subcontractors UT Austin, ASU, and CU Boulder to develop **ALPACA** (Autonomous Learning with Probability & Abstraction for Competency Awareness), a general framework for competency-aware autonomous agents, particularly those based on reinforcement learning (RL). ALPACA provides insight into RL agent competencies and empowers users to examine and constrain agent behavior, facilitating trust building with human teammates and dramatically improving safety for real-world applications.

An ALPACA-enabled autonomous agent can:

- Communicate its tasking strategies and expected performance in natural language
- Identify the (observable and hidden) conditions that affect its behavior
- Assess its behavior and tasking outcomes in specific scenarios
- Quantify its confidence, both in its tasking performance and its competency assessments
- Update users when its competency has changed or may breach competency boundaries
- Adapt its behavior to better maintain performance and conform to user expectations

ALPACA communicates competency through two means:

1. **General competency statements** describe the strategies, performance, and behavioral conditions of the agent that have been observed previously.
2. **Specific competency assessments** predict the strategy and performance of the agent in a specific scenario, both pre-mission and online. These assessments are responsive to user interests, can address novel scenarios, and can be updated online.

In order to achieve the goals of the DARPA CAML program, the Draper ALPACA team developed the following key technical advancements:

- **Condition identification** through decision tree learning over procedurally generated human-interpretable features, both directly observable and hidden. Achieves DARPA's *coverage* requirements.
- **Structured-language strategies** based on temporal logic inferred over abstracted and partitioned trajectory data. Achieves DARPA's *correctness* requirements.
- **Probabilistic world models (PWMs)**, based on recurrent deep generative modeling, that accurately forecast agent states over long time-horizons while quantifying both aleatoric and epistemic uncertainty. Achieves DARPA's *fidelity* requirements.
- **Event-triggered online outcome assessment** that leverages PWMs to assess and re-assess in real-time the agent's competency in a specific scenario. Achieves DARPA's *reliability* requirements.

The Draper ALPACA team studied, demonstrated, and evaluated these advancements on two simulation-based RL application systems: a Pusher Robot manipulation task and a UAV flight in variable weather task. Both in-house and third-party verification and validation showed that the team was able to achieve all of DARPA's target metrics for the CAML program.

2 Objectives

Competency awareness is critical to establishing calibrated levels of trust between autonomous agents and users. Users need to understand what the autonomous agent can do under different conditions, how it will do it, and how likely it is to succeed. Calibrated trust accelerates the learning of how to use and work with a system and avoids system misuse and underuse. Reinforcement learning (RL) autonomy is a particularly important application for competency awareness due to its black box nature, potential for catastrophic failure, and sometimes surprising emergent behavior.

A user operating or working with an autonomous agent will typically want to be able to ask the following questions, which we divide into questions about *general* competencies and *specific* competencies:

	General Competencies (Demonstrated)	Specific Competencies (Forecasted)
Performance Assessment	How successful is the agent in different tasks?	How well is the agent predicted to perform this task?
Behavioral Assessment	What behaviors does the agent use to accomplish tasks?	What is the agent forecasted to do?
Explanation	What conditions affect the performance and behavior of the agent?	Why are this performance and behavior predicted? What are the sources of uncertainty?

ALPACA should answer these questions about competency in real-time, allowing the autonomous system to be used confidently and correctly, even by an inexperienced user. ALPACA should also detect and alert when any user-defined competency boundaries are in danger of being crossed, enabling increased safety and predictability when operating the system.

ALPACA should achieve DARPA's target competency metrics as follows:

1. *Coverage*: ALPACA should identify all significant conditions that affect the agent's performance and behavior.
2. *Correctness*: ALPACA should accurately predict and describe the strategies used by the agent to achieve the task.
3. *Fidelity*: ALPACA should assess calibrated probabilities of agent tasking success and other outcomes of interest to the user.
4. *Reliability*: ALPACA should update its competency assessments given new information and provide a mechanism to allow the autonomous agent to adapt in real time to maintain user competency expectations.

In addition to achieving these quantitative metrics, ALPACA should produce competency statements and assessments that are as interpretable, informative, and actionable as possible.

3 Challenges

Each component of the ALPACA competency framework comes with its own challenges:

Competency	Component	Challenges
General Competency Statements	Strategies	• Separating continuous agent behavior into discrete strategies • Deciding which information is important and unimportant to communicate • Scaling strategy models to complex systems with many behaviors • Predicting strategies correctly in chaotic or stochastic systems • Inferring temporal logic descriptions efficiently • Extracting features that are both interpretable and explanatory to describe strategies
	Conditions	• Crafting human-interpretable conditions • Formulating all conditions and combinations of conditions in their simplest format • Identifying hidden (unobservable) conditions • Explaining the full diversity of agent behavior with discrete conditions • Finding conditions that predict both strategy and performance • Ensuring conditions are generalizable when they may not be causal
Specific Competency Assessments	Probabilistic World Models	• Accurately quantifying both aleatoric and epistemic uncertainty • Untangling aleatoric and epistemic uncertainty • Enabling forecasting over long time horizons • Evaluating very high-dimensional probability models
	Outcome Assessment	• Reducing the computational burden of accurate assessments • Deciding when to best update an outcome assessment • Communicating the correct level of granularity
	Reliability	• Determining when competency boundaries might be crossed • Feeding back competency assessment information quickly

4 Methodology

4.1 Simulated Application Systems

We developed ALPACA to be generic and applicable to a wide variety of RL applications. Thus, we wanted to ensure that we evaluated our research on several distinct applications with varying challenges. We performed most of the development and study on two simulated RL problems: one in robotic manipulation and one in UAV flight in variable weather conditions.

4.1.1 Pusher Robot

The "Pusher Robot" application, shown in Figure 1, is a PyBullet simulation of a 2-DoF robot arm that manipulates a ball within a walled pen to various target locations. The agent has full observability of the system, which is represented as a 12-DoF state plus 2-DoF task specification (goal location for the ball). The ball can vary in size/density and the ball, arm, and target are randomly initialized in different valid starting configurations.

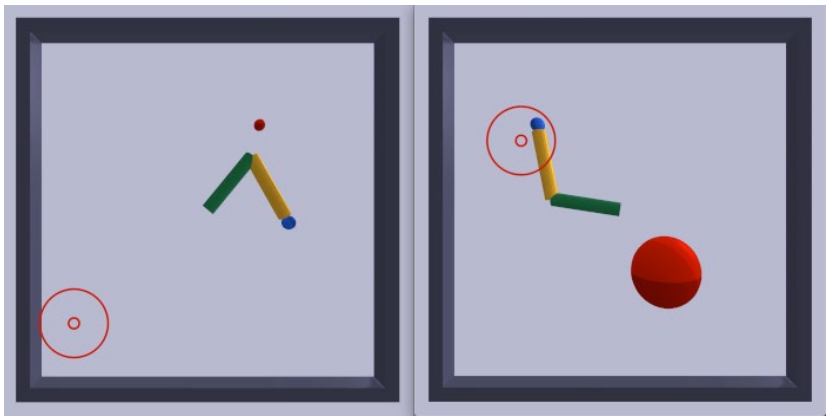

Figure 1: Visualization of two different random starting states for the Pusher Robot

The specific task for the RL agent is to move the ball to the target within 200 timesteps (10 seconds). The two primary failure modes for the agent are to knock the ball out of its own reach or to take too long to accomplish the task. The agent is rewarded for minimizing the distance between the ball and the target and can use any part of the robot arm or environment walls to help accomplish its task. There is no single obvious strategy for most initial configurations, which affords the agent creativity in solving the task.

The Pusher Robot provides a good balance between complexity and simplicity for competency-awareness research. The task is relatively challenging (with a well-trained agent achieving only ~90% success) and the agent behaviors are varied and sometimes surprising. Although the Pusher Robot is a simplification of real-world robotic use-cases, it contains interesting collision interactions between the ball, agent, and environment that generalize to a wide array of potential ALPACA applications. In addition, the geometry of this environment creates interesting behavior symmetries (reflections and 90-degree rotations) that are important for ALPACA to be able to detect.

The Pusher Robot RL system itself is a feed-forward, fully connected neural network policy model distilled from training data produced by model-predictive control (PDDM planner) within a learned recurrent neural network (RNN) world model of the system. The policy takes a representation of the initial state and target as input and outputs the amount of torque to apply to

the two robot arm joints at the current time step. The distilled policy model is used instead of the original planner since it is much faster, deterministic, and allows for easier V&V.

4.1.2 UAV in Gazebo

Our second simulation application system involves controlling a UAV to goal locations in variable weather (wind and temperature) and carrying a varying payload mass. The UAV simulation is built in Gazebo, an open-source 3D dynamic robotic environment. In addition to modeling and running our UAV platform, we leveraged Gazebo's high-fidelity modeling of terrain, lighting, and various man-made structures to create realistic mission scenarios. Much of our customization takes the form of Gazebo plugins, which are shared libraries loaded into Gazebo which enable fine grained control of most aspects of the simulator.

Figure 2 depicts our Gazebo plugin, which models the platform dynamics, the platform battery, and environmental winds. All communication to and from the plugin is done via ROS messaging. Upon start, the plugin pauses Gazebo and initializes the models with initial UAV state s_0. We follow a *pause-simulate-pause* convention to ensure even time steps: starting in a paused state, when an action a_t is received, the plugin applies that action to the UAV model, then resumes and simulates Gazebo for k steps, then pauses Gazebo and returns the resultant state s_{t+1}.

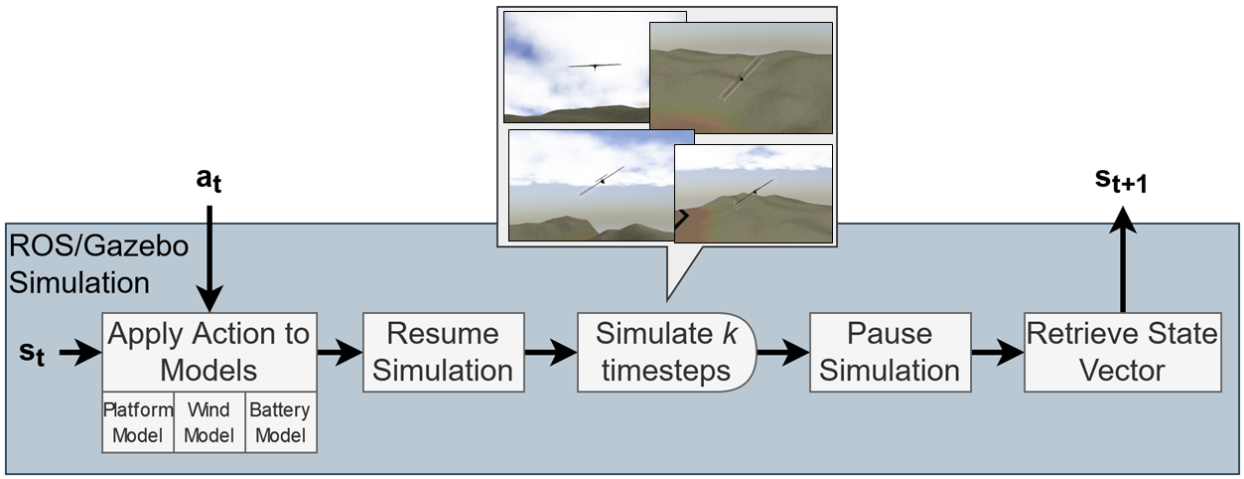

Figure 2. Block diagram of our UAV Gazebo plugin. The plugin begins by pausing Gazebo. When an action a_t is received, the plugin applies that action to the models at state s_t then simulates for k steps. The plugin next pauses the simulation and returns the new state s_{t+1}.

The UAV platform is modeled as a small, fixed wing Techpod UAV from the ROS RotorS package with a continuous state and action space. The four-dimensional continuous action space is composed of control surface deflections of the ailerons, elevator, rudder, and thrust. The 18-dimensional continuous state space is composed of the 6d vehicle pose, the 6d vehicle pose derivatives, the 3d local wind vector, the battery level, the local temperature, and the payload mass. All these variables are randomized at the start of a trajectory. The environmental and platform effects (wind, temperature, and payload) all impact the difficulty of the task and the trade-off between the tasking and the battery life. Once the battery life has reached 0, the UAV can only glide without applying any thrust. The RL agent's task is to reach the randomly selected goal locations within 60 seconds without crashing.

The RL agent itself uses an RNN world model with a simple planner that selects actions in a discretized version of the 4-dimensional action space. The world model predicts the *change* of the vehicle state at each time step in a coordinate frame centered on the UAV and oriented along the vehicle's heading. The reward function for the agent rewards decreasing the distance between the UAV and the goals, lightly penalizes battery usage, and strongly penalizes flying too close to the ground. The agent can successfully fly to at least one of two random goals within 60 seconds roughly 70% of the time.

The UAV Gazebo simulation and RL agent are further detailed in our Ref. [8].

4.2 ALPACA System Design

To cover the variety of different questions that users might want answered about agent competency, ALPACA communicates competency in two ways:

1. **General competency statements** based on a decision tree that predicts strategies and performance from conditions. These statements are generated directly from real trajectory data.
2. **Specific competency assessments** that use a probabilistic world model (PWM) to forecast strategies and outcomes from specific initial conditions. These assessments are generated from predicted trajectories, and do not directly rely on having relevant real trajectory data.

The overall ALPACA system, shown in Figure 3, is based on a model-based RL framework, through the agent is not required to use model-based RL. The following sections describe the ALPACA components that enable the communication of these two types of competencies.

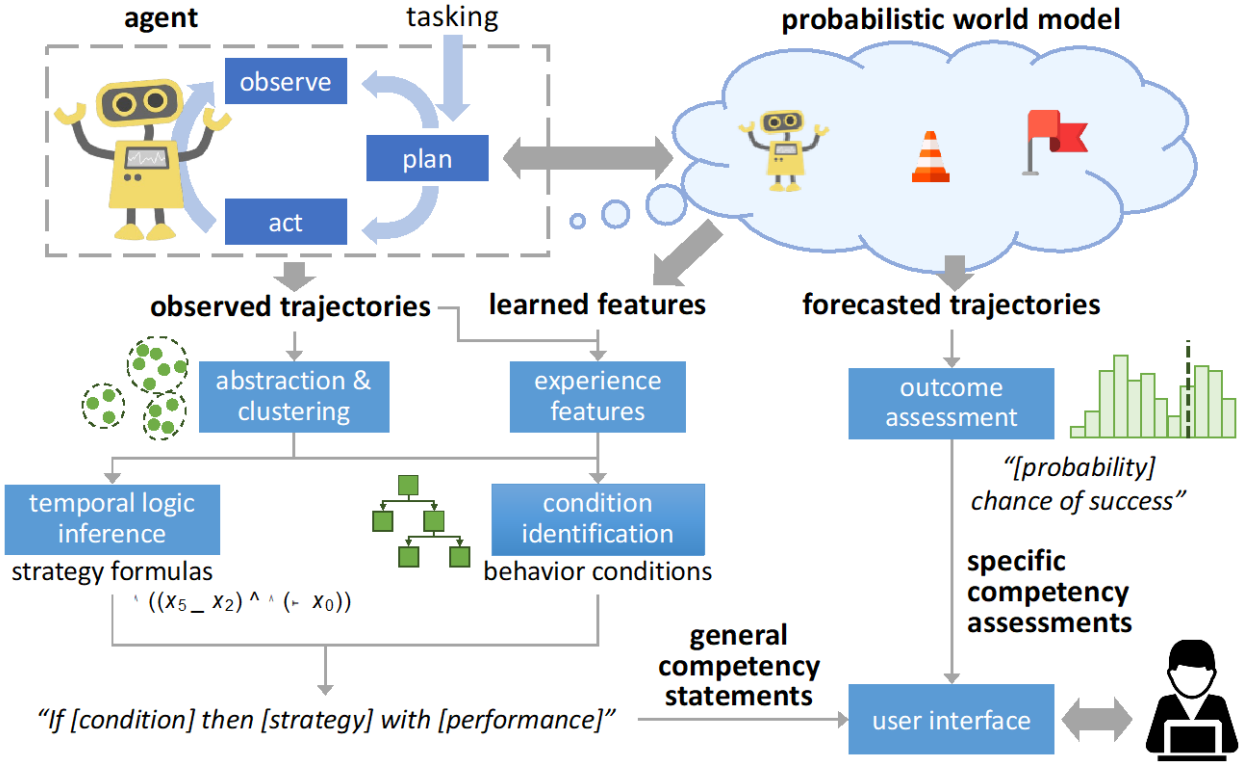

Figure 3: Overall ALPACA system diagram leading to general competency statements and specific competency assessments.

4.3 Probabilistic World Models

The key enabler of ALPACA's specific competency assessments is the probabilistic world model (PWM), which probabilistically forecasts trajectories for a given initial scenario and an agent policy. ALPACA uses these forecasted trajectories to perform assessment of various outcome metrics, which gives the user expectations for the agent's behavior and performance prior to or during operation. In addition, the forecasted trajectories themselves can be visualized to better allow the user to understand how the agent will accomplish its tasking.

PWMs are designed to predict the next states of the agent-environment system given the actions. The actions can result from any policy in response to tasking from the human user. To formalize the description of the PWM, we first define a trajectory as a sequence of state action pairs:

$$\mathcal{T} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{t-1}, \boldsymbol{s}_t)$$

where $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ represent state and action at time t. The PWM, parametrized as a probabilistic recurrent neural network (RNN), is trained on a collection of trajectories with random actions. The PWM learns to approximate the following probability distribution:

$$p(\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_T | \boldsymbol{s}_0; \boldsymbol{a}_0, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{T-1})$$

where T indicates the time horizon of interest to predict out to. This distribution assumes that only one observation is received from the true environment (denoted as $\boldsymbol{s}_0$) and the rest of the states must be predicted without receiving any other observations.

For complex autonomous agents, such as in robotics applications as we are considering for ALPACA, the agent behavior observed through its states are highly multi-modal, non-Gaussian, non-Markovian, and highly correlated both temporally and spatially. Some example trajectory distributions derived from a stochastic version of our Pusher Robot application are shown in Figure 4. The non-Gaussian distributions observed in the shown state dimensions (due to the highly non-linear dynamics of the system) are indicative of the uncertainty that is experienced by the autonomous agent.

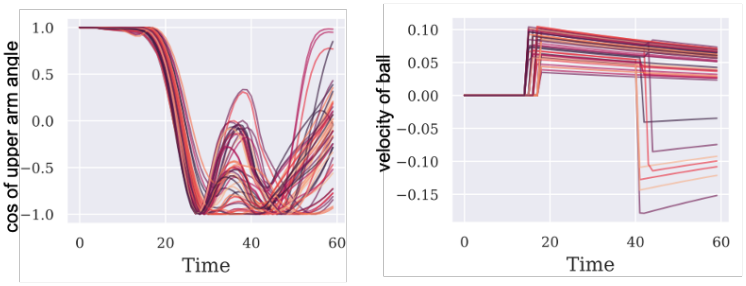

Figure 4: Examples of two state dimensions from the Pusher robot that show complex behavior patterns. PWM must be able to capture such distributions while forecasting the agent's states.

The uncertainties encountered by an autonomous agent can be separated into *aleatoric* or *epistemic* based on their properties and sources:

- *Aleatoric uncertainty* arises from the data due to factors such as process noise, measurement noise, hidden parameters, and/or any other existing randomness in the environment. It is an irreducible form of uncertainty.

- *Epistemic uncertainty* arises from the model due to insufficient training data, out-of-distribution testing, and/or any errors during the modelling process. It is a reducible form of uncertainty.

ALPACA PWMs simultaneously account for both types of uncertainties, as illustrated in Figure 5. To study our ability to accurately quantify aleatoric and epistemic uncertainty together, we formulated stochastic versions of the simulation environments that are more representative of the real world and incorporate explicit aleatoric uncertainty. Our basic approach is to first quantify epistemic uncertainty and then to learn a model of aleatoric uncertainty that accounts for any epistemic uncertainty of the model in the training dataset.

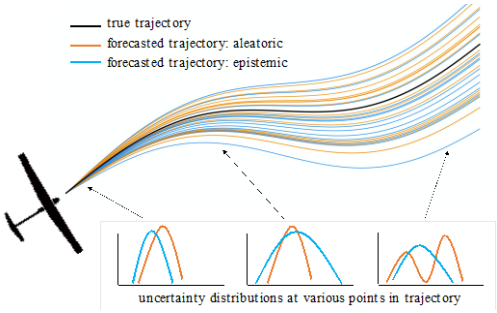

Figure 5: Probabilistic trajectory forecasting for a single agent with aleatoric (orange) and epistemic (blue) uncertainties

Epistemic uncertainty arises from the modelling process and hence relates to the parameters of the neural network world model. To quantify this type of uncertainty, we must place a distribution over the model parameters. While there are various methods to achieve this, deep ensembles [17] is a well-established and validated technique. To formulate deep ensembles, we use a collection of RNN world models. Each of the individual RNN models is trained to learn output the most likely next state $\boldsymbol{s}_{t+1}$ given current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$ and its memory of the previous states and actions. This model is sufficient to capture the long-horizon dynamics and uncertainty of a deterministic fully observable environment. Each RNN is trained with the same collection of data but with random weight initializations and random sampling of the batches during training. This training approach approximates a distribution of the learned model parameters given the training data, thus capturing the epistemic uncertainty in the probabilistic model ensemble.

4.3.1 Aleatoric Uncertainty Quantification

While epistemic uncertainty quantification is sufficient for assessing competency in deterministic and fully observable simulation environments, aleatoric uncertainty must be quantified as the complexity of the world increases. This is especially true for real world environments which are more likely to be noisy and lack full observability. As a result, much of the research in ALPACA for uncertainty quantification focused on developing new methods for aleatoric uncertainty quantification over long time horizons. Since most of the existing literature focuses on epistemic uncertainty and/or model aleatoric uncertainty as Gaussian distribution, there was a need to formulate novel methodologies, especially as we go into applications that are non-Gaussian, non-Markovian, multi-modal, and highly correlated both spatially and temporally.

We use deep generative models to quantify aleatoric uncertainty because of their ability to approximate high-dimensional joint probability distributions directly from data. ALPACA PWMs are based on a conditional version of the variational autoencoder (VAE) [16] architecture, which introduces probabilistic latent variables $\boldsymbol{z}$ to capture the unobserved or stochastic features of the data. The encoder part of the autoencoder maps predicted states onto this latent representation and the decoder maps latent representations onto predicted states. Both the encoder and decoder are conditioned on the previous state and action. Since the PWM must model dynamic systems, we use a recurrent version of the conditional VAE to capture the full temporal information, see our Ref. [3].

Introduction of recurrency in this way allows a clean separation of the temporal states from one timestep to the next despite their non-Markovian property. Following the methods of original VAE loss derivation using Jensen's inequality and Evidence Lower Bound formulation [16], we achieve the following objective criteria for our conditional recurrent VAE PWM:

$$\mathcal{L}(\boldsymbol{s}_{1:T}|\boldsymbol{s}_0;\boldsymbol{a}_{0:T-1}) = -\sum_{t=0}^{T-1}\mathbb{E}_{q_\phi(\boldsymbol{z}_t|\boldsymbol{s}_{0:t};\boldsymbol{a}_{0:t})}[\log p_\theta(\boldsymbol{s}_{t+1}|\boldsymbol{s}_{0:t};\boldsymbol{a}_{0:t};\boldsymbol{z}_t)]$$
$$+\beta\ \mathrm{KLD}[q_\phi(\boldsymbol{z}_t|\boldsymbol{s}_{0:t};\boldsymbol{a}_{0:t})||p_\theta(\boldsymbol{z}_t)]$$

where ϕ represents the encoder parameters, θ represents the decoder parameters, q_ϕ represents an approximate distribution over the latent variable, $\mathbb{E}$ represents the expectation operator, and KLD represents the KL-divergence function. The two terms on the right-hand side are known as the reconstruction loss and the KL-divergence loss, respectively. The β parameter controls the interplay of these two terms and should be tuned to prevent posterior collapse.

The graphical model for the derived recurrent-VAE model and final model structure are shown in Figure 6. The encoder and decoder components can be separated into the inference model and generative model portions as shown. Both the inference and the generative models are used during the training phase, but only the generative model that samples from the learned latent distribution is used during test time to predict the next state.

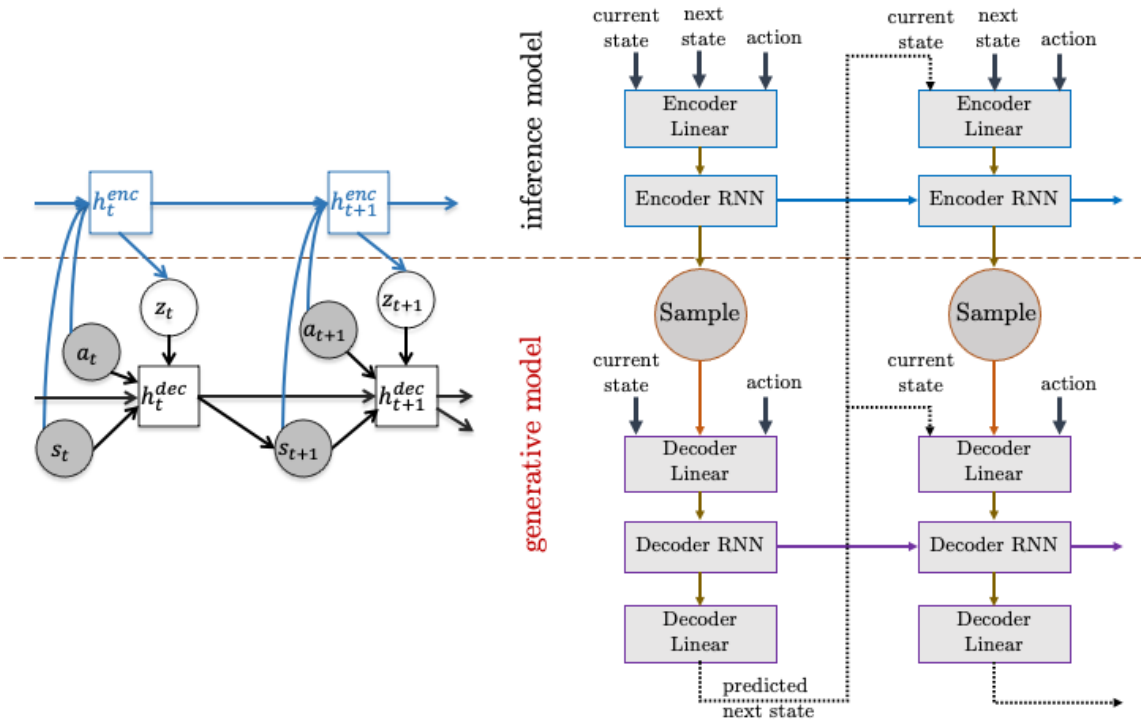

Figure 6: View of the graphical model and the final architecture for our developed recurrent-VAE model. Both components show the recurrency, input states and actions, and the latent variables.

In Ref. [2], we further detail our generative modeling aleatoric uncertainty method and provide an evaluation of the PWM outcome probability calibration compared to baseline methods. We compare outcome probabilities using the Brier score, which is a proper scoring rule. The Brier

score values for both the stochastic Pusher Robot application and stochastic UAV Gazebo application are shown in Table 1. As a baseline, we also show results from another probabilistic neural network (NN) that outputs mean and variance of a Gaussian distribution at each time step. This model is representative of existing state-of-the-art methods for performing aleatoric uncertainty quantification that assume Gaussian and Markovian environments. These results clearly show the effectiveness of our generative method.

Table 1 Brier score results for the stochastic Pusher environment and UAV environment (lower is better).

	Recurrent VAE	Baseline Probabilistic NN
Stochastic Pusher	**0.072**	0.391
Stochastic UAV	**0.190**	0.272

Finally, we developed a method for training a generative model of aleatoric uncertainty while accounting for epistemic uncertainty in the training data, detailed in Ref. [1] and illustrated in Figure 7. By quantifying epistemic uncertainty in the training set using the ensemble of RNN models, we can subtract out the effect of the epistemic uncertainty on the observed errors when training the aleatoric uncertainty conditional recurrent VAE. This avoids the double counting problem that occurs when training errors are not always dominated by aleatoric uncertainty.

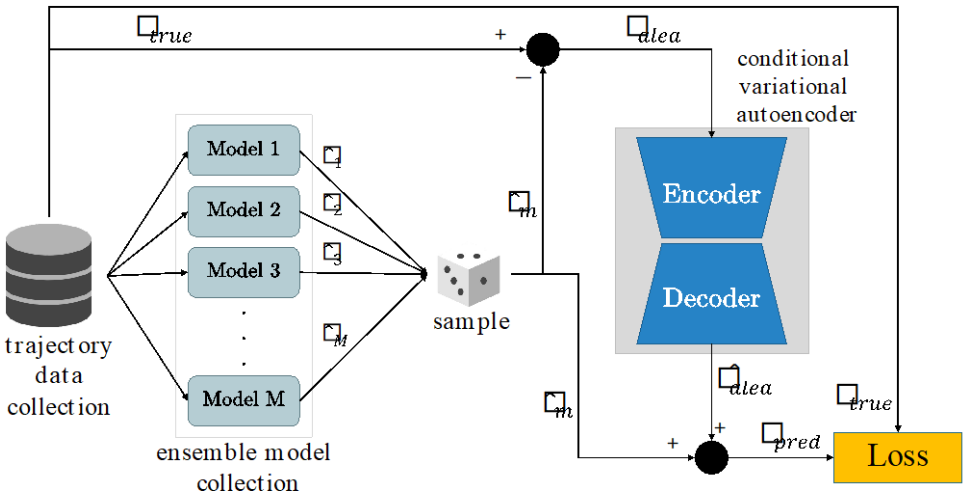

Figure 7: Our method of training a conditional VAE to model aleatoric uncertainty while disentangling epistemic uncertainty.

4.4 Strategies

Agent strategies are extracted unsupervised from raw trajectory data and describe *how* the agent accomplishes its tasking to a human user. First, the trajectories are abstracted into higher-level, human-interpretable features. Then, related features are grouped into strategy categories that correspond to different aspects of the agent behavior. Finally, strategies within strategy categories are described using temporal logic formulas, a structured language format more interpretable by humans, over the previously identified abstract features.

4.4.1 Trajectory Abstraction

Since raw trajectory data is very high-dimensional and high rate, trajectory abstraction is a key step in the inference of human-interpretable strategies. An abstract trajectory is made up of a

temporal sequence of feature vectors that are designed to be simple and human interpretable. A single abstract trajectory can potentially represent many raw trajectories since it describes them at a lower level of granularity. Trajectory abstraction improves the interpretability of individual trajectories, identification of high-level strategies, and generalizability of strategy prediction.

The first step of the abstraction process is identifying the appropriate abstract features and time granularity for the abstracted trajectories. Abstract features describe the behavior of the trajectory at a given moment in time or time interval. Figure 8 show an example of this process. The left depicts the raw state vectors, and the right shows the corresponding abstract trajectory. Colors in the raw state vector correspond to the colors of the three abstract feature states.

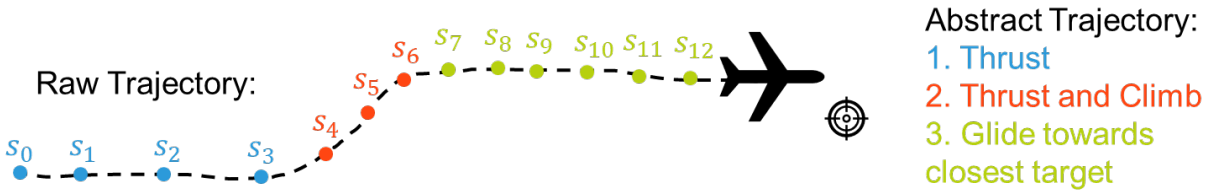

Figure 8 Example correlation between a raw trajectory and an abstract trajectory.

In general, good abstract trajectory features have the following four properties:

	Property	Description	Quantification
1	Human-interpretable	A user should be able to understand what the feature represents, confirm the behavior from observing a trajectory, and find the information relevant to the tasking	None, must be proposed by a human
2	Contextual	A feature should describe as much of the behavior as possible with respect to the tasking context	Number of state/task dimensions in used in calculation
3	Relevant	The features should not contain information that is insignificant to the observed behavior	Invariance under detected behavior symmetries
4	Deliberate	Features should represent varied but predictable behavior rather than incidental effects from chaos/randomness	Mutual information of feature with initial state

The crafting of abstract trajectory features cannot be fully automated since human-interpretability is difficult to quantify. Instead, ALPACA relies on an expert to *propose* potential features that it then evaluates under quantitative properties #2, #3, and #4. For example, Pusher Robot trajectories were broken down into contact events where the ball interacted with either the robot or environment. Human-proposed abstract features for these contact events included: which part of the environment or robot the ball contacted, the time length of contact (bounce or push), the momentum change from the contact, and the arm rotation prior to and during the contact. The abstract trajectories are typically much shorter than the raw trajectories. The time

granularity for the abstracted trajectory is set to the lowest resolution at which the selected features maintain their variation so relevant behavior information is not lost.

The ALPACA assessment of how *contextual* a feature is from how many dimensions from the raw state and tasking information it is based on. For example, a feature like "distance between UAV and closest goal" is a better abstract trajectory feature than "UAV altitude" since it includes much more of the tasking context and more aspects of the agent behavior.

Next, the abstract feature should communicate *relevant* information rather than information that is unimportant or redundant. One way to define relevant information is that it is invariant to changes in perspective. To identify equivalent perspectives for an RL system, we must identify the behavior symmetries of the system. We do this automatically using the machine learning symmetry detection pipeline shown in Figure 9 and detailed in our Ref. [9]. The pipeline transforms the raw trajectory dataset under given candidate transformations and uses a discriminator neural network to try to distinguish between the original trajectories and the transformed ones. If the discriminator cannot distinguish between the original trajectories and the transformed ones, the transformation describes a symmetry of the system. These discovered symmetries give insight into the physical environment and the agent behavior. For example, in the UAV application, we find that translations in x and y are symmetries for the agent (the behavior is indistinguishable) but translations z are not since the agent tries to avoid crashing into the ground. We use all these symmetries together to craft an invariant coordinate representation for UAV that allows for the generation of many relevant features.

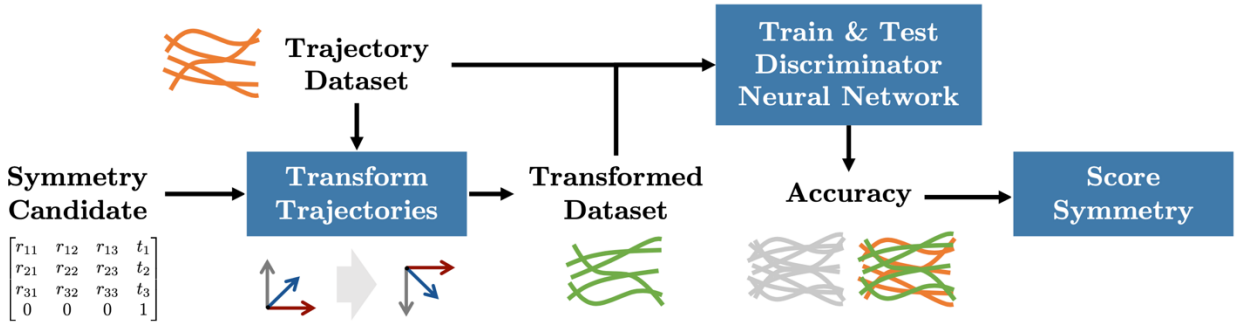

Figure 9: ***Symmetry detection*** *– ALPACA automatically identifies the (approximate) symmetries in trajectory data to aid the abstraction process. Symmetry detection uses a "discriminator" neural network to determine if there is a meaningful difference between the original trajectories and trajectories under a given transformation. These symmetries are then used to craft more "relevant" abstract trajectory features. See our Ref. [9] for more.*

Finally, to test if a feature represents *deliberate* variation, we calculate the mutual information between it and the initial raw state. We use this metric to find agent strategies that are predictable based on the initial conditions of the environment, rather than arising through randomness or chaos. We want features (x) with the greatest mutual information ($I(x; s_0)$) with the initial state (s_0). A more intuitive way to understand how this metric satisfies our objective is to consider the relationship between mutual information and entropy in the following equation:

$$I(x; s_0) = H(x) - H(x|s_0)$$

The mutual information is the difference of two entropies: the uncertainty of the feature x and the uncertainty of the feature x conditioned on the initial state s_0. This can be interpreted as how much the uncertainty in the feature is reduced by knowing the initial state. Using this metric, we will discard features that do not vary much (low entropy) or are unpredictable (high conditional

entropy). We will be left with features with higher but more predictable variation. $H(x|s_0)$ is challenging to compute since the initial state s_0 is typically continuous and high dimensional and thus the frequencies of x occurrences in the data cannot be easily binned in its representation. Instead, we train a neural network to estimate the probabilities of abstract features occurring as a function of the initial state and the abstract trajectory up to that point in time. Once the probabilities are estimated, it is straightforward to compute the required entropies.

All three quantitative metrics (to test if features are contextual, relevant, and deliberate) are used to down select and prioritize abstract trajectory features for creating strategy categories.

4.4.2 Strategy Categories and Clustering

Next, ALPACA breaks down agent behavior into a small set of independent *strategy categories* that describe different aspects of agent strategy based on the selected abstracted trajectory features. Each strategy category contains two or more mutually exclusive strategies. Strategy categories are based on related subsets of the selected abstract trajectory features and should be mostly *uncorrelated* with each other. For example, the four strategy categories for the UAV application relate to its strategies in the following behavioral aspects:

1. How the agent uses rotational maneuvers
2. How much altitude change vs. horizontal change the UAV experiences
3. How the agent applies thrust or glides
4. How the agent selects which goal to fly to next

Strategy categories are essentially a dimensionality reduction of the abstract trajectory features. To achieve this, we calculate the correlation between different abstract features, to allow us to group relatively independent sets of them.

Finally, ALPACAS clusters the abstracted trajectory data, broken down into representations corresponding to each strategy category, into two or more strategy clusters using k-means. One of the biggest challenges in clustering is defining a distance metric over abstracted trajectories. We use three primary distance metrics: a dynamic-time-warping distance metric, an edit-based distance metric, and a length invariant "stretch" distance metric. The dynamic-time-warping distance metric uses the Hamming distance with dynamic time warping to best align two trajectories and adds a penalty for warping that must be done. The edit-based distance metric is like the Levenshtein distance used for text string matching and uses the Wagner-Fischer algorithm to calculate the minimum cost for using insertions, deletions, and substitutions to match two abstracted trajectories. The stretch distance stretches out the shorter of the two trajectories so that it is if the longer of the two trajectories and uses linear interpolation to determine values. We typically found that the "stretch" distance provided the most intuitive results. Once clustered, the abstract trajectory clusters feed into ALPACA's temporal logic strategy inference component to describe the strategy clusters in human-interpretable terms.

4.4.3 Strategy Temporal Logic Inference

For ALPACA, we developed a series of algorithms for strategy inference in terms of temporal logic formulas. Temporal logic statements capture temporal, logical, and/or spatial relations over the execution of a system in the abstract trajectory features [10][11][20]. Since the abstract trajectory features are human interpretable, the temporal logic statements can be translated into natural language to communicate agent strategies with a human user.

We investigated several variants of temporal logic, including ones based on linear temporal logic (LTL) and signal LTL (SLTL), a version of LTL in which the statements are interpreted against sequences of rational-valued variables rather than merely binary variables. Additionally, we investigated LTL as a binary classification problem (find formula describing boundary between two data samples), unsupervised classification (perform simultaneous clustering and inference), and one-class classification (describe one data sample). The strategy inference tools in the ALPACA toolbox are:

Tool	Description	Use
Binary classification LTL	Find the formula that separates two trajectory sets based on binary predicates	Binary abstract trajectory
Binary classification SLTL	Find the formula that separates two trajectory sets with binary and continuous predicates	Continuous and binary abstract trajectory features
Unsupervised LTL	Find the formula that best clusters the given trajectory sample based on homogeneity gain	No clear clusters
One-class LTL	Find the most informative formula that describes all the trajectories	No clear clusters and "unconditional" strategies

In general, our strategy inference tools attempt to find the simplest strategy formula that describes the abstract trajectory data to a required level of accuracy. They are also typically robust to noise, see our Refs. [11] and [12], unlike standard approaches to temporal logic. The algorithms based on binary classification allow us to describe individual clusters in our strategy categories. Since we can use SLTL in addition to LTL, our abstract trajectory features can be binary, categorical, or continuous. Even when the clustering step has failed to find clearly distinct clusters, our unsupervised LTL based on simultaneous clustering and temporal logic inference allows us to extract separated strategies.

In addition to performing inference over clusters of trajectories, the one-class LTL inference allows ALPACA to describe "unconditional" strategies that always (or nearly always) hold true. For example, when applied to the UAV application, this approach produced the following unconditional strategies, illustrated in Figure 10: "either the UAV always glides, or it never glides" and "a change in yaw angle is always accompanied by a change in roll angle." Our one-class LTL algorithm also allows us to control how *specific* a formula is. In general, good strategy formulas have a balance between specificity and simplicity (interpretability).

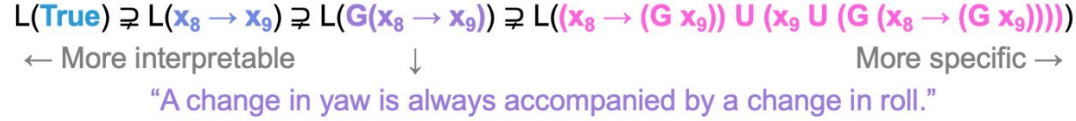

Figure 10: A sequence of statements generated by the one-class LTL algorithm, where "L" stands for the language of the temporal logic statement, "G" stands for the temporal logic operator "always," and U stands for the temporal logic operator "until."

Though our unsupervised and one-class LTL algorithms do not require clustered trajectory data, they do still require a quantification of distance between abstracted trajectories. The unsupervised LTL algorithm requires this in order to cluster similar trajectories with each other and the one-class LTL algorithm requires it to define how specific a formula is (how tight the decision boundary is) around the data. We use the same distance metrics previously described in the clustering section for these algorithms.

While temporal logic inference does have scalability challenges, the trajectory abstraction and strategy categorization process mitigate this by compressing the trajectory feature representation (for each strategy category) and shortening the lengths of the trajectories (by reducing the time resolution). Since many trajectories will appear as duplicates in these abstracted representations, the inference only needs to be performed over a small set of "unique" trajectories to be generally applicable. In these representations, a dataset of 10,000 trajectories can usually be condensed into several dozen unique abstract trajectories for strategy inference within each strategy category.

4.5 Behavioral Conditions

ALPACA general competency statements are based on conditions identified using a decision tree trained to predict strategies and outcomes from semantic *experience features*. Extracting these experience features is thus key to high coverage and correct general competency statements.

4.5.1 Experience Features

Experience features are derived from observable elements at the start of a mission: the initial state, the task parameters, the scene, and environmental measurements, see our Ref. [4]. While these individual elements may be few (10s-100s), the combinatoric experience features derived from them is often enormous (10,000s of features). We developed a geometric reasoning engine that procedurally generates experience features from the meaningful geometric elements (points, line segments, planes, circles, spheres, boxes, vectors) in each scene. For the Pusher Robot, these geometric elements (in 2D) included the robot arm segments, the ball center, the ball edge, the robot hand, the robot elbow, the origin, the target center, the target edges, and the walls and pen corners labeled by what they were closest or furthest to. The experience features included not only the distances between all these elements but also comparisons between distances of these elements. For example, one binary generated Pusher Robot experience feature was:

- "Distance(target center, origin center) < Distance(target edge, ball center)"

Two continuous generated Pusher Robot experience features were:

- "AbsAngle(target center, ball center)"
- "Distance(ball center, wall center closest to target)"

Altogether, there were roughly 40,000 Pusher Robot experience features. While most of these automatically derived experience features do not significantly affect system behavior, the decision tree is able to identify the ones that best correlate with observed strategies and outcomes.

ALPACA experience features also include non-geometric features derived from other observable variables (ball mass, temperature, payload mass). Good experience features tend to follow the same properties of good abstract trajectory representations: they should be human-interpretable, contextual, and relevant.

4.5.2 Hidden Experience Features

One challenge is when agent behavior or performance depends on an experience feature that is unknown or unobservable. We term these "hidden" experience features. Many systems require the ability to operate effectively in similar but slightly varying environments. For example, when controlling a robot to move through an environment, it must be able to operate on surfaces with different coefficients of friction. Like friction, many conditions that affect system behavior are fixed over a long period of time, perhaps even the entire course of a trajectory, but may vary from one trajectory to the next. These "hidden parameter" conditions can affect the behavior and performance of an RL system. In many applications, it may not be clear how many hidden parameters exist or how they affect the system dynamics. We developed an unsupervised method to learn feature representations of trajectories by their disentangled hidden parameter values. This feature space can then be used in downstream classification or analysis tasks to interpret the hidden parameters.

Our approach leverages the fact that an RNN world model handles partially observable state representations by incorporating a memory of previous state observations and actions for its predictions. The features learned in this world model memory should include information on any hidden parameters mixed in with other transient and irrelevant information.

We developed two innovations that enable the learning of a disentangled feature space to represent hidden parameters, detailed in our Refs. [18] and [19]. Our first innovation is to modify the world model training algorithm to constrain part of the world model's internal recurrent representation (memory) to be time-invariant: the model's predictions should not change when the values of the time-invariant features are replaced with their corresponding values at another time-step along the same trajectory. This constraint essentially prohibits the world model from storing transient information in the time-invariant features, leaving only information pertaining to the hidden parameters.

Our second innovation is a metric learning approach to map the time-invariant features into a space with a meaningful metric that we term a latent bisimulation metric. While the time-invariant features contain only hidden parameter information, it is not possible to directly relate two sets of time-invariant features. For example, two sets of time-invariant features may have very different values but have a similar effect on the predictions produced by the world model and thus represent a similar set of hidden parameters. The only way to relate two sets of time-invariant features in a semantically meaningful way is through the usage of the world model. To fix this, we learn a mapping from the time-invariant feature space into a metric space with distance proportional to the difference in system behaviors. This is akin to learning a bisimulation on the latent state (i.e., the hidden parameters) as opposed to the observable state.

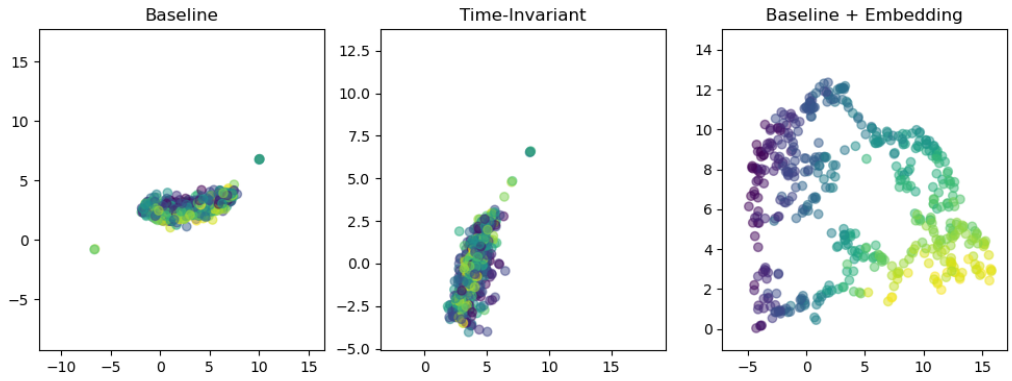

Figure 12: Visualization of UAV trajectories with temperature parameter hidden.

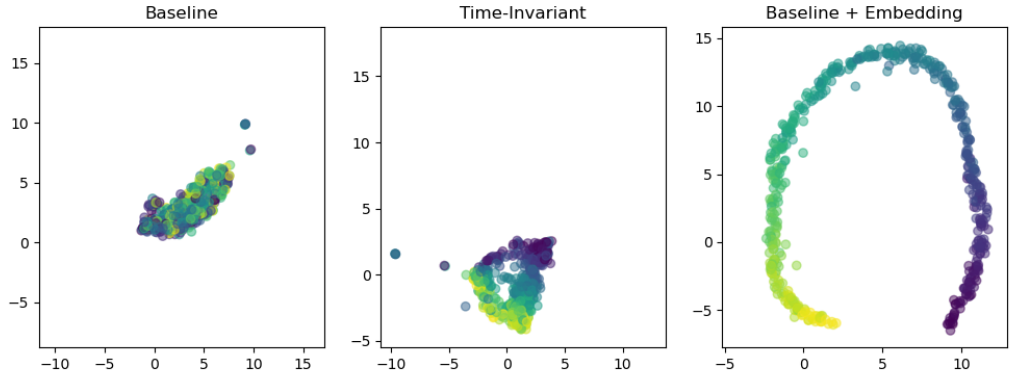

Figure 12: Visualization of UAV trajectories with payload parameter hidden.

We create artificially hidden parameters in the UAV application system by removing variables from the observable state: the payload mass and the ambient temperature. We then use our algorithms to learn feature spaces representing the values of those hidden parameters. Figure 11 and Figure 12 show our results for the temperature and payload mass parameters, respectively, visualized in two dimensions. Each marker represents a trajectory, where the color indicates the value of the hidden parameter. It is clear that for both hidden parameters, the embedded feature space organizes the trajectories by hidden parameter value effectively. Interestingly, for the hidden payload mass, the embedded feature space maps the trajectories onto a 1D manifold. This is ideal as there is only one hidden parameter. While the feature space for the hidden temperature does not recreate the desired 1D manifold structure, it does effectively organize the trajectories by parameter value.

Hidden experience features are included with the rest of the observable experience features when generating competency statements.

4.6 General Competency Statements

To construct general competency statements, ALPACA must predict agent strategies and performance from an interpretable combination of experience features. Given a trajectory training dataset, ALPACA extracts the experience features from the initial state and the strategies from matching the learned temporal logic formulas to the full trajectories. Then, ALPACA trains decision trees on the experience features to predict the observed strategies, one for each strategy

category. These decision trees are highly limited in their depth and number of branches, so they remain human interpretable. An example decision tree is shown in Figure 13 with the distribution of strategies in each leaf shown as a pie chart.

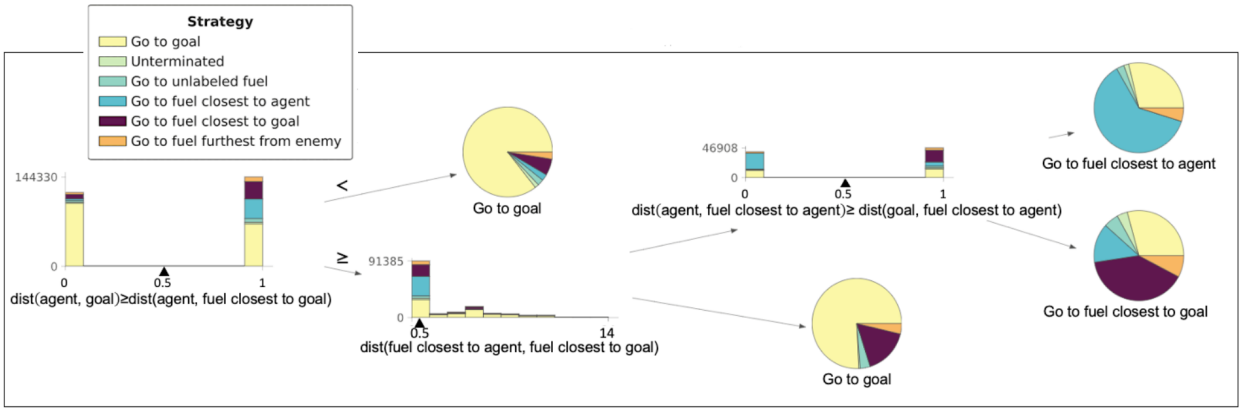

Figure 13: Example decision tree from a simple "grid world" environment studied in Ref. [4]

Each leaf in the decision tree corresponds to a general competency statement of the form:

- "If [condition] then [strategy] with [performance]."

For example, one of the Pusher Robot competency statements we generated was:

- "If *ball radius < 2 cm* then *tap ball* with *85% success*.

However, sometimes the identified conditions are much more complex, such as:

- If *Dist(target center, origin) < Dist(target edge, ball edge)* and *ball radius > 3 cm* and *Dist(target center, ball center) > 20 cm* then *bounce ball off wall* with *12% success*.

There can be several different competency statements that correspond to a single strategy, particularly successful and unsuccessful versions of that strategy. The performance statistic in the competency statement is calculated from the average performance in the trajectories that match its condition. Since it is rare that the decision tree can perfectly predict the observed strategy, ALPACA general competency statements are probabilistic.

When the real trajectory datasets are limited, ALPACA can use the PWM to "imagine" trajectory data to ascertain the agent behavior in regions where data is sparser. This capability relies on the PWM's ability to generalize across different trajectories and even across different agent tasks. We know it can do this since it is able to generalize from random trajectory data to a learned policy without additional training. The further from the training domain that the imagined trajectories are generated, the higher epistemic uncertainty the trajectories will have, which will be reflected in the confidence of the competency statement.

4.7 Specific Competency Assessments

Competency assessments are specific to a given scenario and complementary to general competency statements. Competency assessment can be performed both pre-mission and online at multiple levels of granularity. ALPACA competency assessments rely on probabilistic forecasts of trajectories using the PWM from a given initial state, task, and scenario context. Example competency assessments might include:

- "52% confident in achieving goal within 15 seconds"

- "85% confident in following glide strategy"
- "Expected total distance traveled: 31-35 meters (90% CI)"

These competency assessments are done both pre-mission and online, while monitoring the user's competency boundaries.

4.7.1 Pre-Mission Competency Assessment

ALPACA's algorithmic and communication framework developed for outcome-based competency assessments is grounded in the Factorized Machine Self-confidence (FaMSeC) paradigm for competency self-assessment in decision-making autonomous agents [14][15] [21]. The main idea of FaMSeC is to compute competency assessments in the form of machine self-confidence statements [13], which reflect the statistical likelihood of an autonomous agent either exceeding or falling short of expected minimal performance bounds on specified task outcomes. The generation of self-confidence statements and associated competency assessments are based on a simple, highly adaptable, and easily interpretable risk-reward measurement process developed by the ALPACA team, called Generalized Outcome Assessment (GOA) [7]. GOA is performed on Monte Carlo samples from the PWM representing the full joint uncertainty of the forecasted trajectory.

GOA weights the forecasted probability of achieving sets of "favorable" vs. "unfavorable" outcomes in the future, where "favorable" implies at or above an expected set of user-defined competency thresholds for each quantifiable outcome measure associated to a task, such that more positive/negative weight is assigned to outcomes which further exceed/fall short of thresholds. Since this approach uses a PWM to simulate full trajectory forecasts and can evaluate arbitrary user-interpretable task outcome metrics, GOA can be readily extended to virtually any autonomous system. GOA automatically incorporates the aleatoric and epistemic uncertainty quantified by the PWM, leading to calibrated assessments even in regimes of high uncertainty.

4.7.2 Event-Triggered Online Competency Assessment

In the presence of uncertainty, competency assessments should be updated over time to account for the most recent observations of the agent state and environment. Additionally, human-robot collaboration is dynamic and users desire to actively engage with autonomous systems to understand them and direct them toward a shared objective. These reasons motivated us to develop online confidence assessment tools. Online assessment allows for a human collaborator to adapt the nature of their collaboration to better fit the task as it evolves over time.

To offset the computational burden of performing repeated simulations in an online competency assessment setting, we developed Event-Triggered GOA (ET-GOA) [8]. ET-GOA significantly speeds up and lowers the cost of GOA re-evaluations with minimal accuracy loss by intelligently determining when new PWM simulations are needed. ET-GOA quantifies the divergence between predicted states up to the current time and actual observed states up to the current time. Any divergence large enough triggers ALPACA to re-assess the agent's competency. In this way, ALPACA only allocates valuable computational resources to re-assessing competency whenever it anticipates that it will be "sufficiently surprised" by a shift previously predicted random state outcomes. The degree of surprise can be formally measured by the statistical distance between previously simulated/predicted state trajectories (generated according to the PWM using previously available information) and the currently observed state trajectory.

To determine when competency may have changed, we leverage the Surprise Index (*SI*) as our statistical distance. *SI* is defined as the sum of probabilities of events more extreme (or less probable) than an observed event given a probabilistic model. *SI* ranges from zero (most surprising) to one (least surprising) and can be thought of as how (in)compatible an observation is with respect to a given model's predictions. We are interested in how "surprising" a real observed state s_t is with respect to the model's state prediction $\hat{s}_t$ at time t:

$$SI(s_t) = \sum_{p(\hat{s}_t)<p(s_t)} p(\hat{s}_t)$$

Observations of an agent's state which are surprising (i.e., unexpected, incompatible) with respect to that agent's existing models and planners can be used to indicate that the agent's competency has changed and may warrant a reassessment. By setting the threshold for the statistical surprise distance to a larger/smaller tolerance, the system will naturally tend to re-assess its competence by re-sampling predicted state trajectories less/more often. Thus, the optimum setting for this triggering threshold should balance computational cost of prediction with ensuring accuracy/reliability in detecting significant dynamic shifts in competency due to changes in previously expected world behaviors and courses of action.

An example of ET-GOA on the UAV application is shown in Figure 14. ET-GOA (green) produces similar results to periodic GOA (blue) despite being much less computationally intensive.

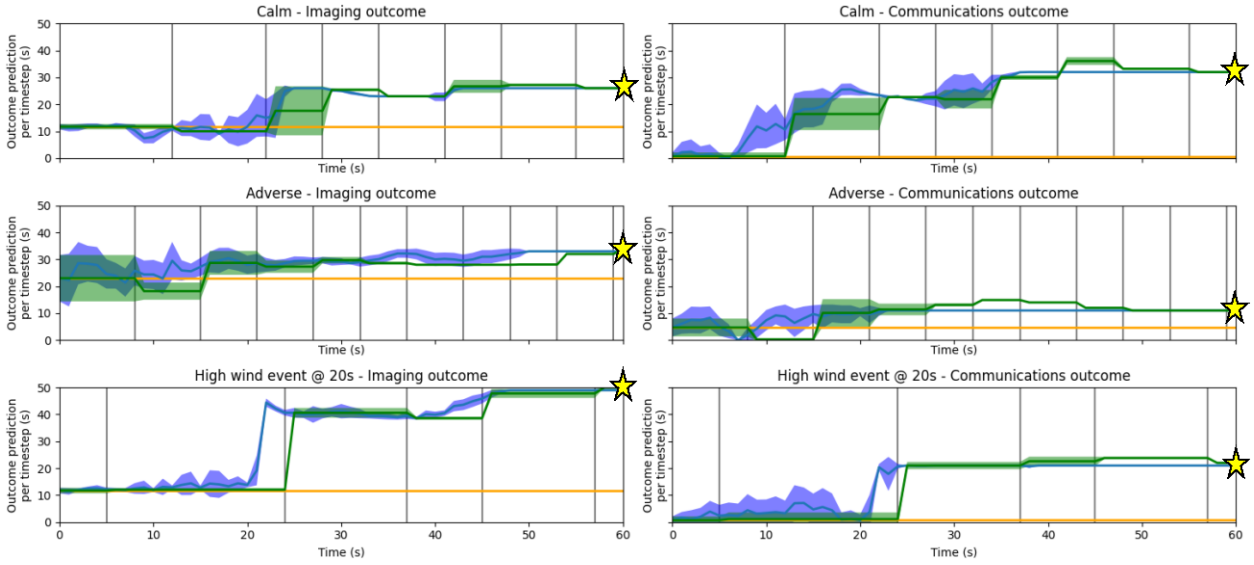

Figure 14: ET-GOA for tasking success in two UAV application tasks (imaging and communications) under high uncertainty and dynamic conditions. The orange line shows the pre-mission GOA mean, the blue band shows (computationally expensive) periodic GOA, and the green band shows ET-GOA. The start shows the actual outcome.

4.8 Reliability Mechanism

One capability that the ALPACA competency tools enable is a reliability mechanism that directly feeds back an assessment into agent decision making. This reliability mechanism allows the agent to better obey its general competency statements or to adhere to user expectations on the fly. Unlike the rest of the ALPACA framework, this mechanism is dependent on the internals of the autonomous agent.

In the case of a model-based RL agent, as we primarily performed our experiments on, it is straightforward to feed competency information directly into the reward function. For example, for the UAV application, we added a term to the planner's reward function that (optionally) encouraged the agent to select actions that followed the forecast strategy. We also added a term to the reward function that added a user-selectable speed minimum for the UAV, with a competency assessment to predict the probability of the speed minimum being obeyed by the agent.

In the case of other RL or autonomous systems where the reward or tasking specification cannot easily be completely changed on the fly, ALPACA's specific competency assessments can potentially be used to select the ideal parameters for the agent policy on the fly given user competency boundaries. This enables competency responsiveness in reaction to user needs.

5 Results

5.1 V&V Methodology

Draper worked in tandem with the government's independent verification and validation (IV&V) team to develop a system that was able to interface with the IV&V test harness and allow the government team to adequately evaluate our ALPACA system. Figure 15 displays an overview of the Draper V&V system and its interfaces with the government system. In CAML Year 3, this V&V process was performed on the UAV Gazebo application system and its ALPACA competency model.

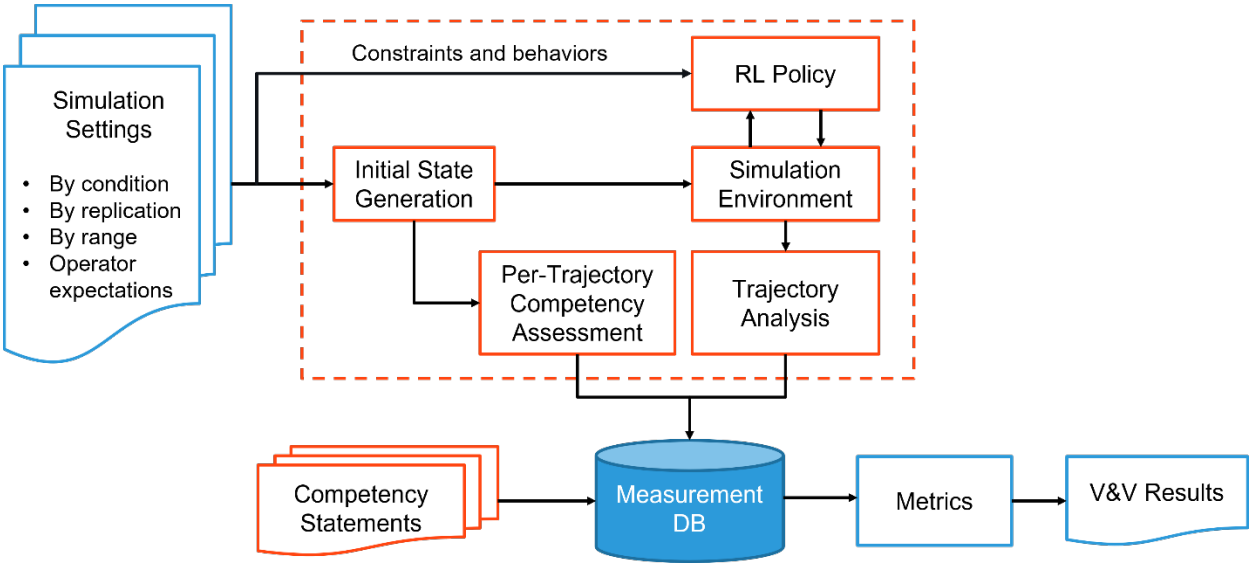

Figure 15 An overview of the Draper verification and validation test harness

There are three modes of operation in which the user can interact with the system–condition, replication, and range. In the condition mode, the user can instruct the system to generate a given number of trajectories for each listed condition. In the replication mode, the user can specify desired replication settings and the system will generate a trajectory matching those settings. In the range mode, the user specifies a range of initial states and desired number of trajectories, and the system randomly samples from the provided range. For all modes, the user can specify operator expectations for the reliability mechanism. These include a behavioral constraint, which specifies that the agent must follow the predicted target strategy, and a behavioral requirement that the agent must stay above a minimum speed threshold.

After the mode of operation is selected, initial states are generated and fed to the Gazebo simulation environment. The operator expectations are fed directly to the RL policy as additional terms to the reward function. Gazebo interacts with the RL policy to generate a full agent trajectory under the user-selected mode of operation.

After completion, the trajectory is analyzed for its strategies, performance, and meeting of expectations. In addition, a unique aspect of our system is a per-trajectory specific competency assessment, which allows us to receive metrics and information about each individual trajectory in addition to the overall assessment provided at the end by the government system. Both measurements are passed via HTTPs to the government IV&V test harness. The harness uses all

the measurements in the database, as well as our competency statements, to calculate the quantitative CAML metrics of correctness, coverage, fidelity, and reliability.

5.2 V&V Results

Draper performed internal verification and validation of our system to assess the correctness, fidelity, and reliability of our system in addition to the results provided by the government team. The Gazebo UAV simulation was new for Year 3 and presented additional challenges when compared to our Year 2 Pusher Robot application. Tiny numerical fluctuations and non-determinism in the simulation can result in large trajectory distances over time. This chaos makes it much easier to predict agent behavior one step ahead instead of many, and it affects coverage, correctness, and reliability validity assessment. The effects of this chaos were mitigated by identifying less chaotic strategies, forecasting probabilistic distributions, and using the reliability mechanism.

Table 2 reports the calculated results. Both correctness and fidelity are calculated by generating 500 trajectories for each of the eight conditions, for a total of 4000 trajectories. Correctness is calculated as the per-inference prediction of agent strategy, and the system reports an average of 90% accuracy across all strategies. This result is comparable to our Year 2 Pusher Robot accuracy of 92% despite the increased complexity of the scenario. Individual strategy accuracies range from 87% in Strategy 7, “UAV does not go to the target with the smallest difference in altitude” to 97% in Strategy 5, “UAV thrusts.” Fidelity is calculated as one minus the Brier score prediction of tasking success per-inference. Again, the score of 90% is equivalent to our Year 2 score of 90%.

As outlined previously, the reliability mechanism can be implemented in two ways—the ability to impose a minimum speed constraint (in this case set at 7 m/s) and the ability to force the system to maintain the predicted target selection strategy. Reliability is tested by generating 500 conditions for combinations of each behavior constraint and requirement, for a total of 1500 trajectories. With the speed constraint off, only 40% of trajectories satisfy it. When turning on this requirement, that percentage jumps to 90% with only a 3% hit to accuracy. When placing the “maintain strategy” constraint on the system, the number of trajectories honoring that constraint go from 84% to 95% with no hit to accuracy. This shows that despite the increased complexities of the UAV Gazebo application, we were still able to introduce a highly effective reliability mechanism.

	Description of Calculation	UAV Gazebo (Year 3)		Pusher Robot (Year 2)
Correctness	Accuracy of per-inference prediction of agent strategy	~90% (87% - 97% per statement)		92%
Fidelity	One minus Brier score for prediction of tasking success per-inference	~90%		90%
Reliability	Conformity to operator expectation with reliability mechanism on	Maintain strategy: 84% (off) → 95% (on) Performance hit: 0.59 (off) → 0.59 (on)	Speed constraint: 40% (off) → 95% (on) Performance hit: 0.59 (off) → 0.56 (on)	N/A

Table 2 The results calculated by Draper for internal V&V

Condition coverage was independently assessed by the government IV&V team. No unidentified conditions were discovered by the government team, though one condition was deemed to have insufficient statistical power. Overall, the ALPACA system achieved all of DARPA's target metrics as determined by the government IV&V process as well as Draper's internal V&V.

6 Conclusions & Future Research

Draper, with UT Austin, ASU, and CU Boulder, developed the ALPACA framework for competency-aware autonomous agents. ALPACA enables black-box agents to communicate their experience and behavior through *general competency statements* and their predicted performance and behavior in a scenario through *specific competency assessments*. By communicating its competency in these ways, an ALPACA-enabled agent can build appropriate levels of trust with human users.

ALPACA achieved the DARPA CAML program's four target metrics through a variety of innovations.

1. *Coverage*: ALPACA identifies the conditions impacting agent behavior and performance by correlating across constructed experience features and extracting learned representations of hidden features.
2. *Correctness*: ALPACA expresses agent strategies in temporal logic reasoning over related groupings of features identified to be human-interpretable, contextual, relevant, and deliberate.
3. *Fidelity*: ALPACA assesses agent outcomes of long time-horizons by using probabilistic world models to simulate trajectories incorporating all sources of uncertainty.
4. *Reliability*: ALPACA re-assess the agent's competency and provides quantitative feedback that can be looped into agent decision-making.

While ALPACA has demonstrated the basic components of competency awareness technology, there are two major areas where significant improvements remain to be made.

First, ALPACA has thus far been demonstrated on simulation applications that can easily generate thousands of training data trajectories. Real-world applications often must operate with limited data and potentially experience out-of-distribution scenarios. Worse, systems could exhibit completely novel behaviors outside of the training dataset, making general competency statements less valuable. Some promising applications of ALPACA can be trained in high-fidelity simulations, but ALPACA must still be able to jump the simulation-to-real gap and appropriately describe any corresponding degradation in system performance. In these applications, specific competency assessments incorporating epistemic uncertainty and out-of-distribution detection are the most promising path forward.

Second, while ALPACA competency statements and assessments are easily machine interpretable, they are not always simple to parse for a human, particularly a non-expert user. In particular, the combinations of conditions identified to predict agent strategies are often complex in order to achieve high prediction accuracy. A major area of improvement is in *user experience*, particularly developing more natural ways of describing, exploring, and interacting with competency statements and assessments. While ALPACA competencies were studied at a fixed level of granularity in the DARPA CAML program, competency awareness tools should enable users to easily understand a high-level view of agent competencies and then drill down into the details of most interest. Future competency awareness work should incorporate a human-focused design and even user studies to further the end goal of building calibrated trust between humans and autonomous agents.

7 References

* Denotes publication from the Draper ALPACA team on the DARPA CAML program

[1] Acharya, Aastha, Rebecca Russell and Nisar Ahmed. "Learning for Forecast Aleatoric and Epistemic Uncertainties over Long Horizon Trajectories." *ICRA* (2023).*

[2] Acharya, Aastha, Rebecca Russell and Nisar Ahmed. "Competency Assessment for Autonomous Agents using Deep Generative Models." *IROS* (2022). https://arxiv.org/abs/2203.12670*

[3] Acharya, Aastha, Rebecca Russell and Nisar Ahmed. "Uncertainty Quantification for Competency Assessment of Autonomous Agents." *ICRA Workshop on Safe and Reliable Robot Autonomy under Uncertainty* (2022). https://arxiv.org/abs/2206.10553*

[4] Acharya, Aastha, Rebecca Russell and Nisar Ahmed. "Explaining Conditions for Reinforcement Learning Behaviors from Real and Imagined Data." *NeurIPS Workshop on Challenges of Real-World Reinforcement Learning* (2020). https://arxiv.org/abs/2011.09004*

[5] Baharisangari, Nasim, Kazuma Hirota, Ruixuan Yan, Agung Julius, Zhe Xu. "Weighted Graph-Based Signal Temporal Logic Inference Using Neural Networks." *IEEE Control Systems Letters (L-CSS)* (2021). https://arxiv.org/abs/2109.08078*

[6] Baharisangari, Nasim, Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu and Zhe Xu "Uncertainty-Aware Signal Temporal Logic Inference." *Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)* (2021). https://arxiv.org/abs/2105.11545*

[7] Conlon, Nicholas, Daniel Szafir, and Nisar Ahmed. "'I'm Confident This Will End Poorly': Robot Proficiency Self-Assessment in Human-Robot Teaming." *IROS* (2022).*

[8] Conlon, Nicholas, Aastha Acharya, Jamison McGinley, Trevor Slack, Camron Hirst, Marissa D'Alonzo, Mitchell Hebert, Christoph, Eric Frew, Rebecca Russell and Nisar Ahmed. "Generalizing Competency Self-Assessment for Autonomous Vehicles Using Deep Reinforcement Learning." *AIAA SciTech Forum* (2022). https://doi.org/10.2514/6.2022-2496*

[9] D'Alonzo, Marissa and Rebecca Russell. "Detecting Symmetries in Trajectory Data using Neural Networks." *AAAI Fall Symposium: Lessons Learned for Autonomous Assessment of Machine Abilities* (2022).*

[10] Feng, Lu, Mahsa Ghasemi, Kai-Wei Chang, and Ufuk Topcu. "Counterexamples for Robotic Planning Explained in Structured Language." *ICRA* (2018). https://arxiv.org/abs/1803.08966

[11] Gaglione, Jean-Raphaël, Daniel Neider, Rajarshi Roy, Ufuk Topcu and Zhe Xu. "MaxSAT-based Temporal Logic Inference from Noisy Data." *Innovations in Systems and Software Engineering* (2022). https://doi.org/10.1007/s11334-022-00444-8*

[12] Gaglione, Jean-Raphaël, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. "Learning Linear Temporal Properties from Noisy Data: A MaxSAT Approach." *International Symposium on Automated Technology for Verification and Analysis (ATVA)* (2021). https://arxiv.org/abs/2104.15083*

[13] Israelsen, Brett and Nisar Ahmed. "'Dave...I can assure you ...that it's going to be all right ...' a definition, case for, and survey of algorithmic assurances in Human-Autonomy trust relationships." *ACM Computing Surveys* (2019). https://arxiv.org/abs/1711.03846

[14] Israelsen, Brett, Nisar Ahmed, Eric Frew, Dale Lawrence, and Brian Argrow. "Factorized Machine Self-Confidence for Decision-Making Agents." (2019). https://arxiv.org/abs/1810.06519

[15] Israelsen, Brett, Nisar Ahmed, Eric Frew, Dale Lawrence, and Brian Argrow. "Machine Self-Confidence in Autonomous Systems via Meta-Analysis of Decision Processes." *HFES* (2019). https://arxiv.org/abs/1810.06519

[16] Kingma, Diederik and Max Welling. "Auto-Encoding Variational Bayes." *ICLR* (2014). https://arxiv.org/abs/1312.6114

[17] Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." *NeurIPS* (2017). https://arxiv.org/abs/1612.01474

[18] Reale, Christopher and Rebecca Russell. "Learning and Understanding a Disentangled Feature Representation for Hidden Parameters in Reinforcement Learning." *AAAI Fall Symposium: Lessons Learned for Autonomous Assessment of Machine Abilities* (2022).*

[19] Reale, Christopher and Rebecca Russell. "Learning a Disentangled Feature Representation for Hidden Parameters in Reinforcement Learning." *ICML Workshop on Human-Machine Collaboration and Teaming* (2022).*

[20] Roy, Rajarshi, Zhe Xu, Ufuk Topcu and Jean-Raphaël Gaglione. "A MaxSAT Approach to Inferring Explainable Temporal Properties," *ICML Workshop on Theoretic Foundation, Criticism, and Application Trend of Explainable AI* (2021).*

[21] Sweet, Nicholas, Nisar Ahmed, Ugur Kuter, and Christopher Miller. "Towards self-confidence in autonomous systems." *AIAA Infotech@ Aerospace Conference* (2016). https://arc.aiaa.org/doi/10.2514/6.2016-1651

[22] Xu, Zhe, Bo Wu, Aditya Ojha, Daniel Neider and Ufuk Topcu. "Active Finite Reward Automaton Inference and Reinforcement Learning Using Queries and Counterexamples," *International IFIP Cross Domain (CD) Conference for Machine Learning & Knowledge Extraction (MAKE)* (2021). https://arxiv.org/abs/2006.15714*

[23] Xu, Zhe, Bo Wu, Aditya Ojha, Daniel Neider and Ufuk Topcu. "Active Automaton Inference for Reinforcement Learning using Queries and Counterexamples," *ICML Workshop on Theoretic Foundation, Criticism, and Application Trend of Explainable AI* (2021).*

[24] Xu, Zhe, Melkior Ornik, A. Agung Julius and Ufuk Topcu. "Information-Guided Temporal Logic Inference with Prior Knowledge." *ACC* (2019). https://arxiv.org/abs/1811.08846