Formal Specification Embeddings for Neuro-Symbolic Autonomy

Beyazit Yalcinkaya
University of California, Berkeley
Berkeley, United States
beyazit@berkeley.edu

Marcell Vazquez-Chanlatte
Nissan Advanced Technology Center
Silicon Valley, United States
marcell.chanlatte@nissan-usa.com

Sanjit A. Seshia
University of California, Berkeley
Berkeley, United States
sseshia@berkeley.edu

ABSTRACT

Formal specifications have long been studied as a means of defining objectives in Reinforcement Learning (RL), largely due to their well-defined operational semantics and compositional nature, which enable correctness guarantees for learned policies. However, most approaches have either been confined to systems trained for a single fixed objective, greatly inhibiting generalization, or required symbolic planning over the induced automata of given specifications, resulting in sub-optimal behaviors. On the other hand, the recent success of foundation models has popularized natural language and demonstrations as expressive instruction modalities. One approach to using these modalities in multi-task RL has been to utilize pretrained text and image embeddings to learn task-conditioned policies. While this approach enables generalization in multi-task systems, it sacrifices any formal correctness guarantees for learned policies. To bridge the gap between the traditional use of formal specifications and more recent techniques that incorporate instruction embeddings in policy learning, in our previous work, we have introduced RAD Embeddings, *provably correct pretrained automata embeddings*. RAD Embeddings distinguish distinct tasks and encode semantic similarities across a large class of temporal specifications. In turn, policies conditioning on RAD Embeddings provide both formal correctness guarantees on their behaviors and optimal behavior across a wide range of tasks. In this paper, we present our approach for pretraining RAD Embeddings and show its use cases in both single-agent and multi-agent RL settings.

KEYWORDS

Reinforcement Learning, Formal Specifications, Representation Learning

ACM Reference Format:
Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2026. Formal Specification Embeddings for Neuro-Symbolic Autonomy. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 6 pages.

1 INTRODUCTION

A key challenge in Reinforcement Learning (RL) is setting objectives that accurately encode the desired task. To this end, *formal specifications* have been proposed as a means for defining tasks for RL policies due to their well-defined operational semantics, concise encoding of long-horizon, temporally extended objectives, and compositional nature [6, 8, 9, 15, 17, 20]. Initial efforts have focused on single fixed objectives, where a given specification is used to define the reward over an augmented state space [14, 15, 18], and correctness guarantees for the learned policies have been given [1, 2, 23]. More recently, conditioning on formal specifications to learn multi-task policies has been studied, but generalization has remained a core challenge [17]. Other efforts have proposed instructing a goal-conditioned policy to follow symbolically computed paths in the automaton representation of a given formal specification [10, 11]; however, these approaches have suffered from sub-optimality due to the *myopia* of their goal-conditioned policies. Overall, prior work has yet to achieve a generalizable, scalable, and optimal incorporation of formal specifications into multi-task RL.

In the broader context of learning-enabled control, natural language [3, 5, 13] and demonstrations [12, 16] have been proposed for learning multi-task policies by utilizing pretrained text and image embeddings. While these instruction modalities provide an expressive and intuitive way to specify tasks to policies, they are inherently ambiguous: natural language can be interpreted in multiple ways, and demonstrations may encode suboptimal or conflicting behaviors. As a result, although these instruction modalities are highly generalizable and scalable, they do not provide formal correctness guarantees with respect to the intended task specification.

To bridge the gap between specification-guided policy learning and recent approaches based on instruction embeddings, in our previous work, we have introduced RAD Embeddings, *provably correct pretrained latent representations of automata induced by formal specifications*[20, 21]. RAD Embeddings are designed to distinguish semantically distinct tasks while capturing meaningful similarities across a broad class of temporal specifications, enabling effective generalization in the space of objectives. As a result, policies conditioned on RAD Embeddings can generalize across tasks without sacrificing optimality. Crucially, since RAD Embeddings are grounded in automata with well-defined semantics, they inherit formal correctness guarantees with respect to the underlying specifications. Moreover, RAD Embeddings are complementary to modern instruction modalities: when formal specifications are available, they can be integrated with text and image embeddings, enabling the principled incorporation of formal task semantics into contemporary learning-based control frameworks.

In the following, we first present the necessary preliminaries, introduce a formal definition of correctness for automata embeddings, and describe our method for learning RAD Embeddings. We then discuss application domains for RAD Embeddings. In *Automata-Conditioned Reinforcement Learning* (AC-RL), they enable policies to generalize across tasks while maintaining optimality. In *Automata-Conditioned Cooperative Multi-Agent Reinforcement Learning* (ACC-MARL), RAD Embeddings facilitate optimal team behavior across a wide range of tasks. Finally, we conclude with a discussion of the implications of this work and outline directions for future research.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

2 PRELIMINARIES

We use **Deterministic Finite Automata** (DFAs) to represent tasks.

Definition 2.1. A **Deterministic Finite Automaton** (DFA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the finite set of states, Σ is the finite alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, where $\delta(q, \sigma) = q'$ denotes a transition to $q' \in Q$ from $q \in Q$ on $\sigma \in \Sigma$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The semantics of a DFA is defined by its final states F and its extended (lifted) transition function $\delta^* : Q \times \Sigma^* \to Q$, where

$$\delta^*(q, \varepsilon) \triangleq q \;\text{ and }\; \delta^*(q, \sigma w) \triangleq \delta^*(\delta(q, \sigma), w).$$

If $\delta^*(q_0, w) \in F$, then $\mathcal{A}$ **accepts** w, i.e., $w \models \mathcal{A}$. If $\delta^*(q_0, w) \notin F$, then $\mathcal{A}$ **rejects** w, i.e., $w \not\models \mathcal{A}$. We assume that once a DFA accepts a prefix, a suffix cannot change the acceptance decision, i.e.,

$$\forall q \in F, \forall \sigma \in \Sigma, \delta(q, \sigma) = q.$$

DFAs can be reduced to a canonical form (up to an isomorphism) through minimization [7], denoted by $\min(\mathcal{A})$. We write $\mathcal{A}_\top$ and $\mathcal{A}_\bot$ for the single-state accepting and rejecting DFAs, respectively. The **progression** of a DFA $\mathcal{A}$ by a word $w \in \Sigma^*$ is defined as:

$$\mathcal{A}/w \triangleq \min\left(\langle Q, \Sigma, \delta, \delta^*(q_0, w), F \rangle\right),$$

i.e., read the word and minimize the DFA.

Given a finite set of DFAs D over some shared alphabet Σ, we define its corresponding **DFA space** $\mathcal{D} \supseteq D$ as follows:

$$\mathcal{D} \triangleq \{\mathcal{A} \mid \exists \mathcal{A}' \in D, \exists w \in \Sigma^*, \mathcal{A} = \mathcal{A}'/w\}, \tag{1}$$

i.e., $\mathcal{D}$ contains all minimized sub-DFAs of D. A DFA space extends a set of DFAs to be closed under random walks. An agent given a task $\mathcal{A} \in D$ must learn to perform all sub-DFAs of $\mathcal{A}$. That is, it must navigate the DFA space $\mathcal{D}$ towards the accepting DFA $\mathcal{A}_\top$. We consider DFA spaces throughout the paper.

Our goal is to learn DFA embeddings that can distinguish distinct tasks for optimal downstream control. To this end, a notion of similarity is needed to compare different DFAs and the tasks they represent. Thus, we proceed by defining **bisimulation** over DFAs.

Definition 2.2. Given two DFAs $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ and $\mathcal{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ over the same alphabet Σ, a relation $B \subseteq Q \times Q'$ is called a **bisimulation** if and only if the following conditions hold:

(1) $(q_0, q_0') \in B$,
(2) $\forall (q, q') \in B.\ q \in F \iff q' \in F'$, and
(3) $\forall (q, q') \in B.\ \forall \sigma \in \Sigma.\ (\delta(q, \sigma), \delta'(q', \sigma)) \in B$.

$\mathcal{A}$ and $\mathcal{A}'$ are said to be **bisimilar**, denoted by $\mathcal{A} \sim \mathcal{A}'$, if B exists.

Bisimilar DFAs have the same language and, consequently, represent the same task. Therefore, one does not need to distinguish between such DFAs when learning their embeddings for downstream learning-based control. Conversely, if two DFAs are not bisimilar, i.e., they represent distinct tasks, then, for optimal control policies, we shall learn embeddings differentiating them.

3 LEARNING AUTOMATA EMBEDDINGS

In this section, we present our recipe for learning **provably correct** DFA embeddings. We first formally state the notion of correctness.

Definition 3.1. An encoder $\Psi : \mathcal{D} \to \mathcal{Z}$, where $\mathcal{D}$ is a DFA space and $\mathcal{Z}$ is a latent space, is said to be **provably correct** if and only if

$$\forall \mathcal{A}, \mathcal{A}' \in \mathcal{D},\ \mathcal{A} \sim \mathcal{A}' \iff \Psi(\mathcal{A}) = \Psi(\mathcal{A}').$$

i.e., the encoder Ψ must guarantee that $\mathcal{A}, \mathcal{A}' \in \mathcal{D}$ are mapped to the same embedding in $\mathcal{Z}$ if and only if they are bisimilar DFAs.

Essentially, the encoder needs to be a **structure preserving map** with respect to bisimulation. Notice that many trivial encoders satisfy this criterion. For example, one can assign unique integer identifiers to distinct DFAs. However, this would require an enumeration of the large DFA space, inhibiting scalability. Furthermore, such an approach would not provide any useful information to the downstream policy about semantic similarities between tasks. Thus, we want to learn such an encoder using function approximation, which would enable us to scale and generalize to a large set of tasks.

We assume that $\Psi : \mathcal{D} \to \mathcal{Z}$ has enough capacity to represent DFAs in its domain, i.e., there exists a parameterization of the learnable encoder that maps distinct DFAs to unique embeddings. However, even under this assumption, such a guarantee does not follow directly, as the claim concerns not only expressivity but also whether the training procedure yields such representations. In the following, we present our approach for learning encoders satisfying the correctness criterion in Definition 3.1. Our method involves training a policy, using the encoder, to solve an **automata bisimulation game**. To this end, we continue with the first ingredient of our recipe: `ReachAvoidDerived` (`RAD`) DFAs, a prior DFA distribution enabling generalization to a large class of tasks.

3.1 `ReachAvoidDerived` (`RAD`) Automata

To generalize to a large class of tasks, we first need to identify a prior DFA distribution $\iota_{\mathcal{D}} \in \Delta(\mathcal{D})$, which defines the DFA space $\mathcal{D}$ in Definition 3.1, and from which we can sample for training the encoder. Recall that we want $\mathcal{D}$ to capture a large set of tasks; therefore, even though it is finite, we do not want to explicitly compute or enumerate it. Thus, we shall define a generator distribution for $\iota_{\mathcal{D}}$ and treat $\mathcal{D}$ as if it were a countably infinite set. However, there is no way to define a uniform distribution over a countably infinite set, meaning that $\iota_{\mathcal{D}}$ has to be biased. Then, we shall find a useful distribution that can help us generalize to tasks of interest.

In the literature, there are two main task classes considered in the context of specification-guided learning-based control. The first class is `Reach` tasks, which specify a partial and/or total order over alphabet symbols defining events in the underlying environment, e.g., "chop the onions and then put them into the pan." The second class is `ReachAvoid` tasks, which also set unrecoverable hard constraints on top of the given ordering tasks, e.g., "chop the onions and then put them into the pan while avoiding collisions." Additionally, in our previous work [20], we have identified two more task classes that might be of interest in certain applications: `Parity` and `ReachAvoidRedemption`. The latter defines `ReachAvoid` tasks with recovery symbols, e.g., "chop the onions and then put them into the pan while avoiding collisions; if you ever collide, then go to the repair station." The former is a subset of the latter, defining parity tasks, e.g., "the light switch must be toggled an even number of times before leaving the room." Our goal is to define a distribution that generalizes to all of these task classes, and potentially to others.

In our previous work [20], we have utilized the graph structure of DFAs to define a general class of tasks called ReachAvoidDerived (RAD) DFAs. These are randomly mutated Reach and ReachAvoid DFAs, defining a richer structure than all the task classes discussed previously. Furthermore, we have empirically shown that task-conditioned policies trained on RAD DFAs generalize to other task classes in both single- [20] and multi-agent settings [22]. Algorithm 1 presents the high-level procedure for sampling RAD DFAs.

Algorithm 1 ReachAvoidDerived (RAD) DFA Sampler

Sample a sequence of one-step Reach and ReachAvoid problems of length $k-1$, where $k \sim$ Uniform and at each step, flip a coin to decide whether to include the Avoid part, call it $\mathcal{A}$.
For each stuttering symbol of $\mathcal{A}$, i.e., a symbol that does not change the state, with probability 0.1, make it Reach or Avoid by flipping a coin; with probability 0.9, keep it unchanged.
$\mathcal{A} \leftarrow \min(\mathcal{A})$
for $i = 1$ to m (where $m \sim$ Uniform) **do**
  $\mathcal{A}' \leftarrow$ Mutate $\mathcal{A}$, i.e., randomly change a transition
  $\mathcal{A}' \leftarrow$ Make accepting states of $\mathcal{A}'$ sinks
  $\mathcal{A}' \leftarrow \min(\mathcal{A}')$
  if $\mathcal{A}'$ is not a trivial DFA (i.e., $\mathcal{A}_\top$ or $\mathcal{A}_\bot$) **then**
    $\mathcal{A} \leftarrow \mathcal{A}'$
  end if
end for
return $\mathcal{A}$

We use RAD DFAs as our prior task generator distribution $\iota_{\mathcal{D}}$, which implicitly defines a DFA space $\mathcal{D}$, to train the encoder Ψ. Next, we continue with the details of our training procedure.

3.2 Automata Bisimulation Game

We learn a provably correct DFA encoder by training a policy to solve an **automata bisimulation game**, presented next.

Definition 3.2. An **automata bisimulation game** is a single-player Markov game defined as $\mathcal{G} = (\mathcal{D} \times \mathcal{D}, \Sigma, T, R, \iota_{\mathcal{D}})$, where

- $\mathcal{D} \times \mathcal{D}$ is the set of states, where $\mathcal{D}$ is a DFA space,
- Σ, the shared alphabet of DFAs in $\mathcal{D}$, is the set of actions,
- $T : \mathcal{D} \times \mathcal{D} \times \Sigma \to \mathcal{D} \times \mathcal{D}$ is the transition function s.t.

$$T(\mathcal{A}, \mathcal{A}', \sigma) = \mathcal{A}/\sigma, \mathcal{A}'/\sigma,$$

- $R : \mathcal{D} \times \mathcal{D} \times \Sigma \to [-2, 2]$ is the reward function s.t.

$$R(\mathcal{A}, \mathcal{A}', \sigma) = r(\mathcal{A}, \sigma) - r(\mathcal{A}', \sigma),$$

where

$$r(\mathcal{A}, \sigma) = \begin{cases} 1 & \text{if } \mathcal{A}/\sigma = \mathcal{A}_\top \\ -1 & \text{if } \mathcal{A}/\sigma = \mathcal{A}_\bot \\ 0 & \text{otherwise,} \end{cases}$$

and

- $\iota^2_{\mathcal{D}} \in \Delta(\mathcal{D})^2$ is the initial state distribution, where $\iota_{\mathcal{D}}$ is a DFA distribution over $\mathcal{D}$, such as the one given in Algorithm 1, and $\mathcal{A}, \mathcal{A}' \sim \iota^2_{\mathcal{D}}$ independently samples a pair.

We solve this game by learning a value function $V : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$. Here, the main idea is to jointly train an encoder $\Psi : \mathcal{D} \to \mathcal{Z}$ to provide embeddings satisfying the criterion given in Definition 3.1. To this end, for $\mathcal{A}, \mathcal{A}' \in \mathcal{D}$, the value function is defined as follows:

$$V(\mathcal{A}, \mathcal{A}') \triangleq \left\| \frac{\Psi(\mathcal{A})}{\|\Psi(\mathcal{A})\|} - \frac{\Psi(\mathcal{A}')}{\|\Psi(\mathcal{A}')\|} \right\|, \tag{2}$$

where $\|\cdot\|$ denotes the l_2-norm. In other words, the value of a game state, i.e., a DFA pair, is defined as the Euclidean distance between the normalized embeddings of the given DFAs. We learn the value function V by applying the Bellman backup at each iteration k as:

$$V^k(\mathcal{A}, \mathcal{A}') \leftarrow R(\mathcal{A}, \mathcal{A}', \sigma) + \gamma\, V^{k-1}(T(\mathcal{A}, \mathcal{A}', \sigma)),$$

where $\gamma \in [0, 1]$ is a discount factor and $\sigma = \pi^k(\mathcal{A}, \mathcal{A}')$ for

$$\pi^k(\mathcal{A}, \mathcal{A}') \leftarrow \arg\max_{\sigma \in \Sigma} \left\{ R(\mathcal{A}, \mathcal{A}', \sigma) + \gamma\, V^{k-1}(T(\mathcal{A}, \mathcal{A}', \sigma)) \right\}.$$

Essentially, we learn a value-based policy that, throughout an episode, generates an evidence string showing that two given DFAs $\mathcal{A}, \mathcal{A}' \in \mathcal{D}$ are not bisimilar, using the representations given by Ψ.

Observe that the value function V, defined in Equation (2), forms a **pseudometric** over the latent space, i.e., it satisfies non-negativity, identity-on-the-diagonal, symmetry, and triangle inequality conditions. This allows Ψ to learn embeddings that enable V to form a distance measure over bisimulation equivalence classes in the latent space. In other words, V forms a **bisimulation metric**, measuring *how bisimilar* two DFAs are and returning $V(\mathcal{A}, \mathcal{A}') = 0$ only for bisimilar DFAs $\mathcal{A} \sim \mathcal{A}'$. In turn, the optimal encoder Ψ^* satisfies the correctness criterion given in Definition 3.1 since $V(\mathcal{A}, \mathcal{A}') = 0$ if and only if $\Psi(\mathcal{A}) = \Psi(\mathcal{A}')$, due to Equation (2). For a full proof of this result, we refer the interested reader to our previous work [21].

We call the latent representations given by an optimal, i.e., provably correct, DFA encoder **RAD Embeddings**, inspired by the RAD task distribution used for training. A provably correct DFA encoder allows us to take a problem defined over a DFA space and, equivalently, reformulate it as a problem over RAD Embeddings. This enables the decoupling of representation and control learning in the training of DFA-conditioned policies for downstream applications, improving sample efficiency, as we showed in our previous work in both single- [20, 21] and multi-agent settings [22]. Before concluding this section, we finally present the details of the encoder.

3.3 Automata Encoder Architecture

Here, we discuss the neural network architecture used for the DFA encoder. With an ethos similar to Section 3.1, we utilize the graph structure of DFAs and employ a **Graph Attention Network** (GATv2), an attention-based graph neural network architecture [4], with a minor modification to include edge features in message aggregation. Below are details of our DFA encoder architecture.

Given a DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, we construct its featurization $G = (V, E, h, e)$, representing it in a graph format, where

- V is the nodes, containing a node for each state of $\mathcal{A}$,
- E is the edges, containing an edge for each transition of $\mathcal{A}$,
- h is the node features, i.e., one-hot vectors encoding whether states are initial, accepting, rejecting, or neither,
- e is the edge features, i.e., one-hot vectors encoding alphabet-based constraints of their corresponding transitions.

We refer to the features of a node $v \in V$ as h_v and the features of an edge between nodes $v, u \in V$ as e_{vu}. In our GATv2 implementation, at each message passing step, node features are updated as:

$$h'_v = \sum_{u \in N^{-1}(v)} \alpha_{vu} W_{msg} \left[h_u \parallel e_{vu} \right],$$

where $N^{-1}(v)$ is the set of nodes with edges to v, W_{msg} is a linear map, and α_{vu} is the attention score between v and u computed as:

$$\alpha_{vu} = \text{softmax}_v \left(a^\top \text{LeakyReLU} \left(W_{atn} \left[h_v \parallel e_{vu} \parallel h_u \right] \right) \right),$$

where a is a vector and W_{atn} is a linear map. For a DFA with n states, we perform n message passing steps, which guarantees that the node representing the initial state of the DFA has received messages from all n nodes. Therefore, we pick the feature vector of this node, i.e., the initial state, as the embedding of the given DFA task $\mathcal{A}$.

Remark 3.3. DFAs are closed under Boolean operations; however, their size, i.e., the number of states, grows exponentially with each composition, e.g., the conjunction of k DFAs each with n states results in a DFA with n^k states in the worst case. Therefore, given a collection of DFAs to be composed under a Boolean operation, we shall avoid explicitly computing the composition. We can easily do so by introducing a syntactic structure for such cases, e.g., if we need to take the conjunction of n DFAs, we can introduce a special **conjunction node** in our graph formulation given above, and utilize the learning capacity of GATv2 to encode such tasks. Our framework can be directly applied in these cases, as the introduction of such **Boolean nodes** is merely a syntactic trick that is independent of the correctness guarantee of the training procedure given in Section 3.2 – see our previous work for more details [20].

4 AUTOMATA-CONDITIONED RL

In this section, we discuss **Automata-Conditioned Reinforcement Learning** (AC-RL) and how RAD Embeddings can be used in this context to improve sample efficiency and scalability. AC-RL is an extension of **goal-conditioned RL** (GCRL). In vanilla GCRL, goals are usually represented as states, or sets of states, specifying goals for the policy to reach by navigating the underlying environment. However, for temporal tasks, i.e., tasks that require reaching a sequence of states, avoiding certain states, etc., using the vanilla GCRL formulation to instruct a goal-conditioned policy to reach sub-goals forming the overall task is inherently **myopic** and therefore results in sub-optimal behaviors. To overcome this myopia, AC-RL represents tasks using DFAs and learns DFA-conditioned policies. This allows agents to take the entire temporal task into account to make optimal decisions in the environment. Below, we formally state the AC-RL problem.

Definition 4.1. Given a Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, P, \iota \rangle$, a DFA space $\mathcal{D}$ over some shared alphabet Σ with a prior distribution $\iota_{\mathcal{D}} \in \Delta(\mathcal{D})$, and a labeling function $L : S \to \Sigma$, an **automata-conditioned policy** is a mapping defined as follows:

$$\pi : S \times \mathcal{D} \to \Delta(A).$$

The **Automata-Conditioned Reinforcement Learning** (AC-RL) problem is to find a policy π maximizing the probability of satisfying a DFA $\mathcal{A} \sim \iota_{\mathcal{D}}$ by navigating the underlying environment $\mathcal{M}$, i.e.,

$$J(\pi) = \mathbb{P}_{\substack{\mathcal{A} \sim \iota_{\mathcal{D}} \\ \tau \sim \mathcal{M}, \pi, \mathcal{A}}} \left[\mathcal{A} / L(\tau) = \mathcal{A}_\top \right], \tag{3}$$

where τ is a trace generated by conditioning π on $\mathcal{A}$ and running it in $\mathcal{M}$. The objective is to solve for the optimal policy $\pi^* \in \arg\max_\pi J(\pi)$, maximizing the probability of satisfying $\mathcal{A} \sim \iota_{\mathcal{D}}$.

AC-RL requires agents to navigate the underlying environment dynamics to reach the accepting DFA $\mathcal{A}_\top$ in the given DFA space $\mathcal{D}$. Notice a key difference between AC-RL and the traditional single-objective formulations for specification-guided RL: in the latter, policies condition on the states of the induced automata, whereas in AC-RL, the policy conditions on the entire DFA. This allows policies to be multi-task and generalize to tasks in the DFA space. However, conditioning on DFAs couples representation and control learning, which can inhibit sample efficiency as we have shown in our previous work [20, 21]. To mitigate this, we reformulate the problem as one over RAD Embeddings rather than as one over DFAs. That is, we learn $\pi' : S \times \mathcal{Z} \to \Delta(A)$, conditioning on the pretrained latent DFA representations given by Ψ^*. Since the provably correct encoder Ψ^* distinguishes non-bisimilar DFAs and such DFAs represent different tasks, π' solves AC-RL optimally. See our previous work for more details and experiment results [19–21].

5 AUTOMATA-CONDITIONED MARL

In this section, we present **Automata-Conditioned Cooperative Multi-Agent Reinforcement Learning** (ACC-MARL) and the use of RAD Embeddings in this domain. In MARL, a key challenge is learning multi-task, decentralized team policies. In such cases, each agent must reason about the tasks assigned to all agents to locally make optimal decisions toward completing both its own task and the team's objectives. In turn, the representation of tasks becomes crucial for generalization and optimal team behavior. To this end, we propose the use of DFAs to represent tasks in this setting, due to their well-defined operational semantics and concise encoding of long-horizon, temporal objectives. Therefore, we continue with the formal statement of the AC-MARL problem next.

Problem 5.1. *Given a Markov game $\mathcal{M} = \langle S, A, P, \iota \rangle$ with n agents, a DFA space $\mathcal{D}$ over some shared alphabet Σ with a prior distribution $\iota_{\mathcal{D}} \in \Delta(\mathcal{D})$, and a labeling function $L_i : S_i \to \Sigma$ for $i \in [n]$, an **automata-conditioned decentralized policy** for agent i employs*

$$\pi_i : S_i^* \times \mathcal{D}^n \to \Delta(A_i),$$

where S_i^* denotes the set of traces of agent i. The joint policy is

$$\boldsymbol{\pi}(\tau, \mathbf{A}) = \left[\pi_1(\tau_1, \mathbf{A}), \ldots, \pi_n(\tau_n, \mathbf{A}) \right],$$

*where $\mathbf{A} \sim \iota_{\mathcal{D}}^n$ indicates an n-tuple of DFAs sampled from the given distribution $\iota_{\mathcal{D}}$. The **Automata-Conditioned Cooperative Multi-Agent Reinforcement Learning** (ACC-MARL) problem is to find a joint policy $\boldsymbol{\pi}$ maximizing the probability of satisfying the conjunction of all DFAs in $\mathbf{A} \sim \iota_{\mathcal{D}}^n$ by navigating the underlying game $\mathcal{M}$, i.e.,*

$$J(\boldsymbol{\pi}) = \mathbb{P}_{\substack{\mathbf{A} \sim \iota_{\mathcal{D}}^n \\ \tau \sim \mathcal{M}, \boldsymbol{\pi}, \mathbf{A}}} \left[\bigwedge_{i=1}^{n} \mathbf{A}[i] / L(\tau_i) = \mathcal{A}_\top \right], \tag{4}$$

where τ is a joint trace, and $\tau_i \in S_i^*$ is a trace of agent i, generated by conditioning $\boldsymbol{\pi}$ on $\mathbf{A}$ and running it in $\mathcal{M}$. The objective is to solve

for the optimal team policy $\boldsymbol{\pi}^* \in \arg\max_{\boldsymbol{\pi}} J(\boldsymbol{\pi})$*, maximizing the probability of satisfying all DFAs in the sampled n-tuple* $\mathbf{A} \sim \iota_{\mathcal{D}}$.

In ACC-MARL, each agent is assigned a DFA task, and the goal is to complete all assigned tasks. This requires agents to condition on each other's DFAs to decide *whether* or *how* to help one another. However, again, conditioning on DFAs couples representation and control learning, a challenge for scalability, as we have previously demonstrated [22]. To overcome this challenge, we utilize RAD Embeddings and reformulate the problem as one over DFA embeddings rather than DFAs, i.e., for each agent i, we learn a decentralized policy $\pi_i' : S_i^* \times \mathcal{Z}^n \to \Delta(A_i)$ conditioning on RAD Embeddings. As RAD Embeddings satisfy the correctness criterion given in Definition 3.1, they can uniquely represent distinct tasks. Therefore, we can safely plug them into the place of DFAs in the problem formulation, while still guaranteeing optimality with respect to Problem 5.1. See our recent work for more details and results [22].

6 DISCUSSION

This work advances the integration of formal specifications into learning-based autonomy by introducing a principled framework for provably correct pretrained automata embeddings. RAD Embeddings reconcile two traditionally competing objectives: (i) strong semantic guarantees inherited from formal methods, and (ii) generalization and scalability properties typically associated with learning-based techniques. By grounding embeddings in bisimulation and learning them through an automata bisimulation game, we obtain task representations that are both semantically meaningful and directly usable for downstream control. As demonstrated in Sections 4 and 5, this decoupling of representation learning from policy learning enables optimality and generalization across large task families without sacrificing correctness.

Beyond the specific applications presented here, RAD Embeddings suggest a broader design pattern for neuro-symbolic autonomy: rather than embedding raw instructions or demonstrations, one can embed formal task semantics and use these representations as a stable interface between symbolic reasoning and continuous control. This perspective clarifies the role of learning in specification-guided control – not as a replacement for formal structure, but as a scalable mechanism for exploiting it.

A natural direction for future work is the deployment of RAD Embeddings in real-world RL and MARL domains. Many practical robotic and autonomous systems already rely on structured task specifications, safety rules, or operating procedures that can be naturally expressed as automata. Integrating RAD Embeddings into such systems would enable multi-task and multi-agent generalization while preserving correctness guarantees that are critical in safety- and mission-critical settings. An important challenge here is interfacing RAD-conditioned policies with partial observability, noisy perception, and hybrid continuous–discrete dynamics, which are pervasive in real-world environments.

Another promising avenue is the inverse problem: learning formal specifications from data. While this work assumes access to task automata, in practice, specifications may be implicit, underspecified, or available only through demonstrations. Recent advances in neural search and generative modeling raise the possibility of synthesizing candidate specifications from data. To the best of our knowledge, exploring whether generative modeling can be used for specification synthesis from data is an open problem. Such models may also shed light on the structure of human-provided tasks and how semantic regularities emerge across task families.

Finally, a scalable solution to the inverse problem would enable a fully end-to-end pipeline: demonstrations, natural language, or other instruction modalities could be mapped to formal specifications; these specifications could then be embedded via RAD Embeddings; and the resulting representations could be combined with text and image embeddings for downstream control. This would allow agents to leverage the expressivity and accessibility of modern instruction modalities while retaining formal correctness guarantees at the level of task execution. We believe combining learning, symbolic structure, and formal semantics is a promising direction for the next generation of general-purpose autonomous systems.

REFERENCES

[1] Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. 2022. A Framework for Transforming Specifications in Reinforcement Learning. In *Principles of Systems Design (Lecture Notes in Computer Science, Vol. 13660)*. Springer, 604–624.

[2] Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. 2023. Policy Synthesis and Reinforcement Learning for Discounted LTL. In *CAV (1) (Lecture Notes in Computer Science, Vol. 13964)*. Springer, 415–435.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. 2024. π_0: A Vision-Language-Action Flow Model for General Robot Control. *CoRR* abs/2410.24164 (2024).

[4] Shaked Brody, Uri Alon, and Eran Yahav. 2022. How Attentive are Graph Attention Networks?. In *ICLR*. OpenReview.net.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong T. Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. RT-1: Robotics Transformer for Real-World Control at Scale. In *Robotics: Science and Systems*.

[6] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. 2020. Deep Reinforcement Learning with Temporal Logics. In *FORMATS (Lecture Notes in Computer Science, Vol. 12288)*. Springer, 1–22.

[7] John Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*. Elsevier, 189–196.

[8] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *J. Artif. Intell. Res.* 73 (2022), 173–208.

[9] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *NeurIPS*. 13021–13030.

[10] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021. Compositional Reinforcement Learning from Logical Specifications. In *NeurIPS*. 10026–10039.

[11] Wenjie Qiu, Wensen Mao, and He Zhu. 2023. Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. In *NeurIPS*.

[12] Juntao Ren, Priya Sundaresan, Dorsa Sadigh, Sanjiban Choudhury, and Jeannette Bohg. 2025. Motion Tracks: A Unified Representation for Human-Robot Transfer in Few-Shot Imitation Learning. In *ICRA*. IEEE, 8802–8810.

[13] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. 2024. Vision-Language Models are Zero-Shot Reward Models for Reinforcement Learning. In *ICLR*. OpenReview.net.

[14] Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, and Sanjit A. Seshia. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *CDC*. IEEE, 1091–1096.

[15] Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A. Seshia. 2025. LTL-Constrained Policy Optimization with Cycle Experience Replay. *Trans. Mach. Learn. Res.* 2025 (2025).

[16] Sumedh Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Biyik, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. 2023. RoboCLIP: One Demonstration is Enough to Learn Robot Policies. In *NeurIPS*.

[17] Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. 2021. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 10497–10508.

[18] Harish Venkataraman, Derya Aksaray, and Peter Seiler. 2020. Tractable reinforcement learning of signal temporal logic objectives. In *Learning for Dynamics and Control*. PMLR, 308–317.

[19] Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit Seshia. 2023. Automata conditioned reinforcement learning with experience replay. In *NeurIPS 2023 Workshop on Goal-Conditioned Reinforcement Learning*.

[20] Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2024. Compositional Automata Embeddings for Goal-Conditioned Reinforcement Learning. In *NeurIPS*.

[21] Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2025. Provably Correct Automata Embeddings for Optimal Automata-Conditioned Reinforcement Learning. In *NeuS (Proceedings of Machine Learning Research, Vol. 288)*. PMLR, 661–675.

[22] Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, Ameesh Shah, Hanna Krasowski, and Sanjit A. Seshia. 2025. Automata-Conditioned Cooperative Multi-Agent Reinforcement Learning. *CoRR* abs/2511.02304 (2025).

[23] Cambridge Yang, Michael L. Littman, and Michael Carbin. 2022. On the (In)Tractability of Reinforcement Learning for LTL Objectives. In *IJCAI*. ijcai.org, 3650–3658.