Proceedings of Machine Learning Research vol vvv:1–15, 2025

Learning Task-Conditioned Policies with Natural Language and Graph Embeddings of Formal Specifications

Beyazit Yalcinkaya BEYAZIT@BERKELEY.EDU
University of California, Berkeley, USA

Marcell Vazquez-Chanlatte MVC@LINUX.COM
California, USA

Sanjit A. Seshia SSESHIA@BERKELEY.EDU
University of California, Berkeley, USA

Abstract

Recent advances in pretrained foundation models have popularized natural language as an instruction modality in Task-Conditioned Reinforcement Learning (TC-RL). While the expressiveness and intuitiveness of natural language make it an appealing medium for task specification, its inherent ambiguity and lack of operational semantics pose challenges in domains requiring assurances. To this end, graph embeddings of formal specifications have been proposed as a complementary medium to natural language embeddings. However, an experimental evaluation of these instruction modalities has been lacking in the literature. In this work, we present an empirical comparison of natural language and graph embeddings of formal specifications for TC-RL. Specifically, we learn policies conditioning on RAD Embeddings, pretrained graph embeddings of formal specifications, and their corresponding language instruction embeddings, obtained from a popular foundation model in the literature, and analyze their performances. To ensure a fair comparison, we introduce a new method that translates tasks given as automata into language instructions and use it to generate task sets for training and testing. Our evaluation shows that RAD Embeddings enable optimal TC-RL, whereas language instruction embeddings suffer from sample inefficiency. Importantly, we show that using these modalities together recovers optimal performance. Our results highlight that RAD Embeddings provide an effective inductive bias for downstream control policies and that they can be combined with other instruction modalities to improve sample efficiency.[1]

Keywords: reinforcement learning, representation learning, formal specifications.

1. Introduction

The proliferation of foundation models has popularized natural language and demonstrations as means for specifying objectives in Task-Conditioned Reinforcement Learning (TC-RL). However, 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 provide semantically rich information to downstream policies, they lack formal correctness guarantees with respect to an intended task specification defining a notion of correctness.

Formal specifications have been proposed as a means for defining objectives for RL policies due to their well-defined operational semantics, concise encoding of long-horizon, temporally extended objectives, and compositional nature (Icarte et al. (2022); Hasanbeig et al. (2020); Jothimurugan et al. (2019); Shah et al. (2025)). In the context of TC-RL, instructing a goal-conditioned policy to

1. Code is available at https://github.com/rad-dfa/rad-vs-instruct.

follow symbolically computed paths in the automaton representation of a given formal specification has been proposed (Jothimurugan et al. (2021); Qiu et al. (2023)); however, these approaches suffer from sub-optimality due to the myopia of their task-conditioned policies. To this end, conditioning on formal specifications to learn task-conditioned policies has been studied, but generalization has remained a core challenge (Vaezipoor et al. (2021)). Overall, previous efforts have yet to achieve a generalizable, scalable, and optimal incorporation of formal specifications into TC-RL.

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, pretrained graph embeddings of automata induced from formal specifications (Yalcinkaya et al. (2024, 2025a)). 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. However, an empirical study comparing RAD Embeddings and pretrained language instruction embeddings in learning TC-RL policies has remained unaddressed.

In this work, we present an empirical evaluation of RAD Embeddings and language instruction embeddings in TC-RL. Our contributions are as follows: (i) we introduce a framework for translating automata to natural language instructions: our method is based on iterative counterexample-guided refinement and uses a Large Language Model (LLM) as an oracle; (ii) we conduct an empirical evaluation of RAD Embeddings and language instruction embeddings in TC-RL across multiple dimensions: latent space structure, sample efficiency, and generalization; and finally, (iii) we show that RAD Embeddings enable optimal task-conditioned policy learning and that they are complementary to language instruction embeddings, i.e., when formal specifications are available, they can be concatenated with other instruction modalities to recover optimal performance, enabling a principled incorporation of formal task semantics into contemporary learning-based control frameworks.

Related Work. Recent approaches has utilized text and image embeddings to use natural language (Black et al. (2024); Brohan et al. (2023); Rocamonde et al. (2024); Hu and Sadigh (2023)) and demonstrations (Ren et al. (2025); Sontakke et al. (2023)) as ergonomic means of task specification in TC-RL. In specification-guided RL, initial work has focused on single fixed objectives, where a given specification is used to define the reward over an augmented state space (Sadigh et al. (2014); Shah et al. (2025); Venkataraman et al. (2020)), and correctness guarantees for the learned policies have been given (Alur et al. (2022, 2023); Yang et al. (2022)). In TC-RL, instructing a goal-conditioned policy to follow symbolically computed paths in automata has been explored (Jothimurugan et al. (2021); Qiu et al. (2023); Jackermeier and Abate (2025); Guo et al. (2025)). However, these approaches remain suboptimal due to the myopia of their policies. To this end, others have proposed conditioning on temporal logic formulas (Vaezipoor et al. (2021)) and automata (Yalcinkaya et al. (2023)); however, generalization to large task classes has remained a challenge. Our recent work has shown that the graph structure of automata allows defining useful priors to facilitate generalization by pretraining automata embeddings and using them for downstream control in both single- (Yalcinkaya et al. (2024, 2025a)) and multi-agent settings (Yalcinkaya et al. (2025b)). To the best of our knowledge, the closest works to our instruction synthesis method from automata are by Castanyer et al. (2025) and Vazquez-Chanlatte et al. (2025); however, the former generates reward functions from prompts, and the latter synthesizes automata from natural language and demonstrations.

2. Preliminaries

Our end goal is to compare language instructions and formal specifications as task representations. Thus, we start background review with **Task-Conditioned Reinforcement Learning** (TC-RL).

Definition 1 *Given a Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, P, \iota \rangle$ and a set of tasks Ψ with a prior distribution $\iota_\Psi \in \Delta(\Psi)$, a **task-conditioned policy** is a mapping $\pi : S \times \Psi \to \Delta(A)$. The **Task-Conditioned Reinforcement Learning** (TC-RL) problem is to find a policy π maximizing the probability of satisfying a task $\psi \sim \iota_\Psi$ by navigating the underlying environment $\mathcal{M}$, i.e.,*

$$J(\pi) \triangleq \mathbb{P}_{\substack{\psi \sim \iota_\Psi \\ \tau \sim \mathcal{M}, \pi, \psi}} [\tau \models \psi], \tag{1}$$

where τ is a trace generated by conditioning π on ψ and running it in $\mathcal{M}$ and $\tau \models \psi$ denotes that the generated behavior τ satisfies the given task ψ. The objective of TC-RL is to solve for the optimal policy $\pi^* \in \arg\max_\pi J(\pi)$, maximizing the probability of satisfying $\psi \sim \iota_\Psi$.

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

Definition 2 *A **Deterministic Finite Automaton** (DFA) is a tuple $\phi = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the finite set of states, Σ is the finite alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ 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 $\delta^*(q, \varepsilon) \triangleq q$ and $\delta^*(q, \sigma w) \triangleq \delta^*(\delta(q, \sigma), w)$. If $\delta^*(q_0, w) \in F$, then we say that ϕ **accepts** w, i.e., $w \models \phi$. If $\delta^*(q_0, w) \notin F$, then we say that ϕ **rejects** w, i.e., $w \not\models \phi$. Throughout the paper, we consider DFAs whose final states are sink states, i.e., $\forall q \in F, \forall \sigma \in \Sigma, \delta(q, \sigma) = q$.*

Next, we list some notation and useful facts about DFAs. DFAs are closed under Boolean operations, such as conjunction, disjunction, and complementation. We write $L(\phi) \subseteq \Sigma^*$ to denote the language of a DFA. We can **minimize** DFAs to a canonical form (up to a state isomorphism) using Hopcroft's minimization algorithm (Hopcroft (1971)), denoted by $\text{minimize}(\phi)$. Two DFAs are called **bisimilar**, $\phi \sim \phi'$, if they are behaviorally identical, i.e., there exists a relation between their states that relates the initial states, preserves acceptance, and is closed under transitions for every input symbol. We write $\phi_\top$ and $\phi_\bot$ for the single-state accepting and rejecting DFAs, respectively.

The **progression** of a DFA ϕ by a word $w \in \Sigma^*$ is defined as:

$$\phi / w \triangleq \text{minimize}\left(\langle Q, \Sigma, \delta, \delta^*(q_0, w), F \rangle\right), \tag{2}$$

i.e., read the word and minimize the DFA. Given a finite set of DFAs Φ' over some shared alphabet Σ, we define its corresponding **DFA space** $\Phi \supseteq \Phi'$ as follows:

$$\Phi \triangleq \{\phi \mid \exists \phi' \in \Phi', \exists w \in \Sigma^*, \phi = \phi'/w\}, \tag{3}$$

i.e., Φ contains all minimized sub-DFAs of Φ'. In other words, a DFA space extends a set of DFAs to be closed under random walks. We consider DFA spaces throughout the paper.

In TC-RL, when tasks are represented as DFAs, we define the given set of tasks Ψ as a DFA space Φ over a shared alphabet Σ. In this case, the operational semantics of DFAs allows us to progress and update the given task to augment the state with the latest minimal DFA representation

of the remaining task. We do so by using a labeling function $\lambda : S \to \Sigma$, mapping environment states to alphabet symbols, and taking the cascade composition of the MDP $\mathcal{M}$ and the DFA space Φ. Specifically, given an action a_t, we first step the environment $\mathcal{M}$ to get the next state s_{t+1}, then progress the task, i.e., $\phi_{t+1} \leftarrow \phi_t/\lambda(s_{t+1})$ as defined in Equation 2, and return the next state along with the latest minimal DFA representation of the task, i.e., (s_{t+1}, ϕ_{t+1}). This allows us to learn Markovian policies for temporal tasks and thus improve sample efficiency – see our previous work on **Automata-Conditioned RL** (AC-RL) for more details (Yalcinkaya et al. (2023, 2024, 2025a)).

We will use language embeddings to efficiently learn instruction-conditioned policies. Similarly, we will use **RAD Embeddings**, provably correct latent DFA representations, to learn automata-conditioned policies. To this end, the following briefly introduces our recipe for learning latent formal specification representations, starting with a notion of correctness for DFA embeddings.

An encoder $\mathcal{E} : \Phi \to \mathcal{Z}$, where Φ is a DFA space and $\mathcal{Z}$ is a latent space, is to be **correct** if $\forall \phi, \phi' \in \Phi,\ \phi \sim \phi' \iff \mathcal{E}(\phi) = \mathcal{E}(\phi')$. i.e., the encoder $\mathcal{E}$ guarantees that two DFAs have the same embedding in $\mathcal{Z}$ if and only if they are bisimilar. This definition is motivated by the fact that bisimilar DFAs are behaviorally identical 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.

We assume that $\mathcal{E} : \Phi \to \mathcal{Z}$ 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. The following presents our approach to learning encoders that satisfy the given correctness criterion. Our method involves training a policy, using the encoder, to solve an **automata bisimulation game**.

Definition 3 *An **automata bisimulation game** is a single-player Markov game defined as a tuple $\mathcal{G} = (\Phi \times \Phi, \Sigma, T, R, \iota_\Phi^2)$ where $\Phi \times \Phi$ is the set of states defined over a DFA space Φ; Σ, the shared alphabet of DFAs in Φ, is the set of actions; $T : \Phi \times \Phi \times \Sigma \to \Phi \times \Phi$ is the transition function defined as $T(\phi, \phi', \sigma) \triangleq (\phi/\sigma, \phi'/\sigma)$; $R : \Phi \times \Phi \times \Sigma \to [-2, 2]$ is the reward function defined as $R(\phi, \phi', \sigma) \triangleq r(\phi, \sigma) - r(\phi', \sigma)$ and $r(\phi, \sigma) \triangleq \mathbb{1}\{\sigma = \phi_\top\} - \mathbb{1}\{\sigma = \phi_\bot\}$; and $\iota_\Phi^2 \in \Delta(\Phi)^2$ is the initial state distribution defined over a DFA space over Φ and $\phi, \phi' \sim \iota_\Phi^2$ samples a pair.*

We solve this game by learning a value function $V : \Phi \times \Phi \to \mathbb{R}$ via iterative Bellman backups. The main idea is to jointly train an encoder $\mathcal{E} : \Phi \to \mathcal{Z}$. Thus, we define the value function as:

$$V(\phi, \phi') \triangleq \left\| \frac{\mathcal{E}(\phi)}{\|\mathcal{E}(\phi)\|} - \frac{\mathcal{E}(\phi')}{\|\mathcal{E}(\phi')\|} \right\|, \tag{4}$$

where $\|\cdot\|$ is the l_2-norm, and a DFA pair's value is defined as the Euclidean distance between the normalized embeddings of the DFAs. See Appendix A and B for details on ι_Φ and $\mathcal{E}$, respectively.

The value function V forms a **pseudometric** over the latent space as it satisfies non-negativity, identity-on-the-diagonal, symmetry, and triangle inequality conditions. This allows $\mathcal{E}$ 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 $V(\phi, \phi') = 0$ only for bisimilar DFAs $\phi \sim \phi'$. In turn, the optimal encoder $\mathcal{E}^*$ satisfies the correctness criterion since $V(\phi, \phi') = 0$ if and only if $\mathcal{E}(\phi) = \mathcal{E}(\phi')$, due to Equation 4. For more details, we refer the interested reader to our previous work (Yalcinkaya et al. (2025a)).

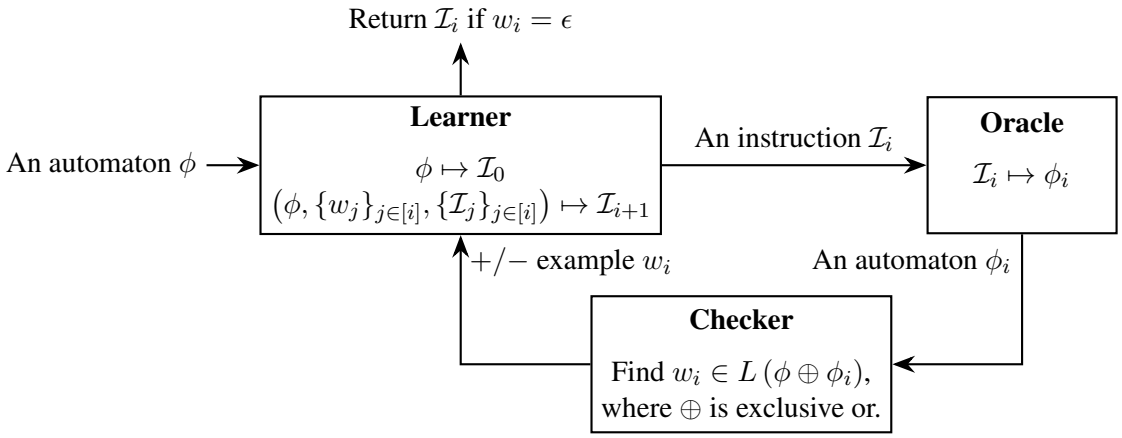

Figure 1: Overview of our method for generating language instructions from automata.

3. Synthesizing Language Instructions from Automata

A key challenge for comparing RAD Embeddings and language instruction embeddings in learning TC-RL policies is assembling a set of tasks with DFA-instruction pairs. To address this, we develop an iterative method for generating such pairs. Our approach combines **Counterexample-Guided Inductive Synthesis** (CEGIS) (Solar-Lezama et al. (2006)) with **Large Language Models** (LLMs) (Brown et al. (2020)). In particular, we use an LLM[2] to map a DFA to an instruction and then map it back to a DFA; if the DFAs are equivalent, then we return the instruction; otherwise, we generate a counterexample to refine the instruction. Our core assumption is that if an LLM can zero-shot translate an instruction back to its original DFA, this provides strong evidence that it encodes the essential structure of the task in a machine-interpretable and semantically precise form.

Figure 1 presents an overview of our approach. Given a DFA ϕ, in the first iteration, we call the **Learner** module, which is an LLM, to generate an instruction, i.e., $\phi \mapsto \mathcal{I}_0$. We then pass the generated instruction $\mathcal{I}_i$ to the **Oracle** module, which is also an LLM, to map it back to a DFA, i.e., $\mathcal{I}_i \mapsto \phi_i$. Consequently, ϕ_i is passed to the **Checker** module to find a counterexample showing that $L(\phi) \neq L(\phi_i)$. **Checker** does so by computing the exclusive or of the original DFA and the given DFA, i.e., $\phi \oplus \phi_i$, and then generating a string from this language, i.e., $w_i \in L(\phi \oplus \phi_i)$. If $L(\phi \oplus \phi_i) = \emptyset$, then this means that ϕ and ϕ_i have the same language, and thus they define the same behavior, meaning ϕ and ϕ_i represent the same task; therefore, we return the instruction $\mathcal{I}_i$. Otherwise, i.e., $L(\phi \oplus \phi_i) \neq \emptyset$, we generate a word $w_i \in L(\phi \oplus \phi_i)$ and augment **Learner**'s input with all previous counterexamples and instructions to generate a new guess, i.e., $(\phi, \{w_j\}_{j\in[i]}, \{\mathcal{I}_j\}_{j\in[i]}) \mapsto \mathcal{I}_{i+1}$.

Our approach closely mirrors CEGIS. **Learner** plays the role of the learning algorithm, proposing a candidate hypothesis based on the current specification and accumulated counterexamples, while the combination of **Oracle** and **Checker** acts as the verification oracle that either proves equivalence or returns a counterexample. Each iteration alternates between hypothesis generation and counterexample-driven refinement. Unlike CEGIS, however, the hypothesis space is the space of natural language instructions rather than symbolic programs. Our method is not guaranteed to terminate, since **Learner** may fail to converge to a consistent instruction. If the procedure does terminate, then it returns an instruction that is zero-shot translatable to a consistent DFA by **Oracle**.

2. We use Qwen3-4B-Instruct by Yang et al. (2025).

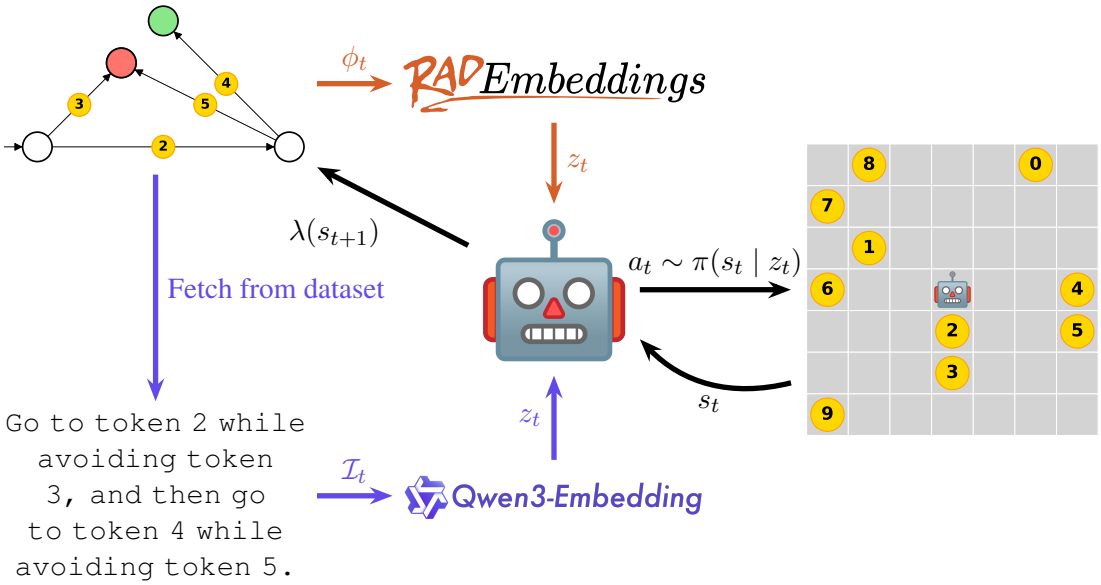

Figure 2: Overview of our pipelines for learning policies conditioned on automata and language.

4. Learning Task-Conditioned Policies with Automata and Instruction Embeddings

We now discuss our approach for learning task-conditioned policies with DFA and language instruction embeddings. To compare RAD Embeddings and natural-language instruction embeddings fairly, we first precompute a dataset of DFA tasks $\hat{\Phi}$. Using the synthesis procedure described in Section 3, we translate each DFA in $\hat{\Phi}$ into a corresponding language instruction, forming a mirrored instruction space $\hat{\Psi}$. For efficient retrieval during training, we hash each DFA and use it as an index into $\hat{\Psi}$, allowing us to quickly fetch the precomputed instruction corresponding to a DFA. We implement two parallel pipelines that differ only in the task embeddings used, ensuring a controlled comparison: one pipeline uses RAD Embeddings, while the other uses 4B model of Qwen3 Embedding, a state-of-the-art foundation model by Zhang et al. (2025). Figure 2 shows our setup.

In the RAD Embeddings branch of the pipeline, a DFA $\phi_0 \sim \hat{\Phi}$ is sampled and encoded via our pretrained encoder to produce a latent embedding $z_0 = \mathcal{E}(\phi_0)$. The policy π' then receives z_t as input and acts in the environment. After executing the action, the environment state s_{t+1} is labeled via λ to progress the DFA, producing an updated DFA $\phi_{t+1} \leftarrow \phi_t / \lambda(s_{t+1})$ that is returned to the policy for the next step. In the Qwen3 Embedding pipeline, we similarly sample a DFA $\phi_0 \sim \hat{\Phi}$ and fetch its corresponding language instruction $\mathcal{I}_0 \in \hat{\Psi}$. The instruction is encoded using the Qwen3 model to produce a latent embedding, which is then passed to the policy. After each action, the progressed DFA is used to fetch the corresponding instruction for the next step. This ensures that the policy always receives an embedding of the instruction that correctly reflects the remaining task and preserves the Markovian structure while conditioning the policy on temporal task information.

By decoupling representation learning from control, both pipelines allow us to study the efficacy of different embedding spaces while keeping all other aspects of policy learning and environment interaction identical. This setup provides a controlled comparison between graph embeddings derived from formal specifications and natural language embeddings derived from instructions.

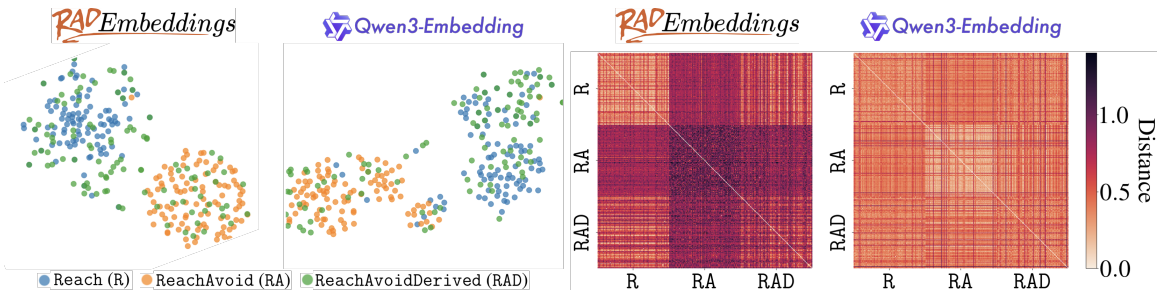

Figure 3: 2-D t-SNE projections (left) and heatmaps of distances based on Equation 4 (right).

5. Empirical Study

In this section, we present an empirical evaluation of DFA and instruction embeddings in TC-RL. To this end, we use the procedure from Section 3 to generate DFA-instruction pairs for three task classes. The first one is `Reach` (`R`) tasks, specifying a partial or total order over alphabet symbols, e.g., "go to tokens 1, 2, and 3, respectively." The second is `ReachAvoid` (`RA`) tasks, which also set unrecoverable hard constraints on top of the given ordering tasks, e.g., "go to token 2 while avoiding token 3." Lastly, `ReachAvoidDerived` (`RAD`) DFAs are randomly mutated `Reach` and `ReachAvoid` DFAs, defining a richer structure than both – see Appendix A for more details on `RAD` DFAs.

We generate 139 pairs for `R` tasks, 479 pairs for `RA`, and 102 for `RAD` tasks. Among these datasets, on average, `R` tasks have 2.8 states, `RA` tasks have 3, and `RAD` ones have 2.6 states. All datasets include a small number of 4-state DFAs, but generating language instructions for DFAs with more states has proven challenging. Thus, most generated pairs are one-step tasks, e.g., " go to token 1 or 9 while avoiding 2 and 8," which requires 3 states. We suspect that with more compute and deploying LLMs with more parameters, it might be possible to translate more complicated DFAs to language instructions. However, we leave the investigation of this matter for future work.

In the following, we tackle three questions:

(Q1) How do DFA and instruction embeddings capture similarities across tasks in latent space?
(Q2) How do DFA and instruction embeddings affect sample efficiency in policy learning?
(Q3) How do policies conditioned on DFA and/or instruction embeddings perform and generalize?

5.1. Latent Space Analysis

We first address **(Q1)** – the short answer is below.

(A1) *DFA embeddings better distinguish tasks both within and across task classes than instructions.*

We sample 100 DFA-instruction pairs from precomputed datasets for `R`, `RA`, and `RAD` tasks. We then compute their corresponding latent representations using RAD Embeddings and Qwen3 Embedding. To understand how tasks are clustered in the latent space, we generate 2-dimensional t-SNE projections of these embeddings. The left of Figure 3 shows that both embeddings cluster tasks in the latent space well. In particular, in both cases, `RAD` DFAs are scattered across the space, overlapping with others. This is expected as `RAD` DFAs are randomly mutated `R` and `RA` tasks and therefore are somewhat similar to these classes. To analyze how task similarities are captured, we compute pairwise distances between embeddings using the pseudometric given in Equation 4. The right-hand side of Figure 3 reports these results, which show that while both embeddings capture similarities within and across task classes, RAD Embeddings show a sharper contrast among tasks.

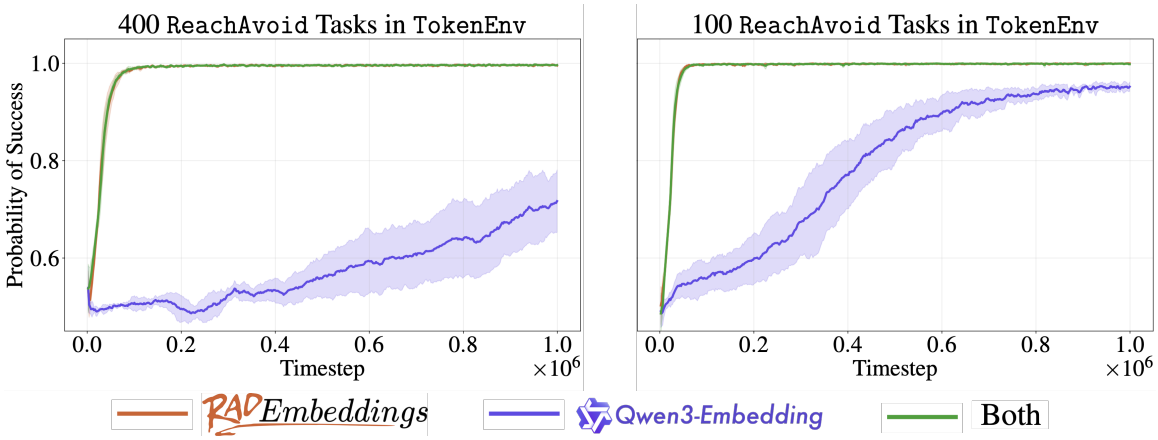

Figure 4: Training curves for learned task-conditioned policies – reported over five random seeds.

5.2. Training Task-Conditioned Policies

Next, we continue with **(Q2)** – the short answer is below.

(A2) *RAD Embeddings greatly improve sample efficiency compared to instruction embeddings.*

We run pipelines presented in Section 4 to train TC-RL policies conditioning on RAD Embeddings and Qwen3 Embedding – see Appendix C for details on policy architecture and hyperparameters. To see whether these embeddings can be combined, we also run a third pipeline concatenating them. We generate two task sets with 400 and 100 pairs, sampled from 479 precomputed `RA` DFA-instruction pairs, to understand the effect of task set size in learning optimal policies. Experiments are conducted in `TokenEnv`, a discrete fully observable MDP. `TokenEnv` defines a grid world, where the agent can move in four cardinal directions or stay at its current cell. There are tokens randomly scattered across the environment. At the beginning of every episode, tasks that involve visiting these tokens are assigned to the agent. See Figure 2 for a visual representation of `TokenEnv`.

Figure 4 presents the results, reporting the success probabilities for TC-RL, as defined in Equation 1, throughout the training; all curves are averaged on five random seeds. On the task set with 100 `RA` tasks, policies conditioned on Qwen3 Embedding approach optimal behavior, indicating that the synthesized language instructions indeed encode meaningful task-relevant information. However, they still fall short of policies conditioned on RAD Embeddings, which consistently achieve optimal behavior. In contrast, on the task set with 400 `RA` tasks, policies conditioned on Qwen3 Embedding exhibit clear sample inefficiency, whereas RAD Embeddings scale seamlessly to this larger task set and retain optimal performance, showing that RAD Embeddings provide a useful inductive bias for scalability. Across both task sets, concatenating RAD Embeddings with instruction embeddings yields performance indistinguishable from conditioning on RAD Embeddings alone. This suggests that RAD Embeddings can be combined with instruction embeddings to get the best of worlds: the precision of formal specifications and the expressive richness of natural language.

5.3. Testing Task-Conditioned Policies

Lastly, we tackle **(Q3)** – the short answer is below.

(A3) *RAD Embeddings enable optimal performance and generalization across task classes.*

Train data; Emb type	`ReachAvoid`	`Reach` (OOD)	`ReachAvoid` (OOD)	`RAD` (OOD)
`RA`-400; RAD Emb	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.94 \pm 0.02}$	0.97 ± 0.01	$\mathbf{0.98 \pm 0.01}$
`RA`-400; Qwen3 Emb	0.72 ± 0.06	0.77 ± 0.05	0.60 ± 0.06	0.74 ± 0.06
`RA`-400; Both	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.94 \pm 0.02}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.98 \pm 0.01}$
`RA`-100; RAD Emb	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.94 \pm 0.03}$	$\mathbf{0.93 \pm 0.01}$	$\mathbf{0.96 \pm 0.02}$
`RA`-100; Qwen3 Emb	0.96 ± 0.01	0.59 ± 0.06	0.55 ± 0.03	0.63 ± 0.03
`RA`-100; Both	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.94 \pm 0.03}$	0.92 ± 0.03	0.95 ± 0.02

Table 1: Test results of policies – reported over five random seeds, each run for 1,000 episodes.

We evaluate the trained task-conditioned policies on both in-distribution and **out-of-distribution** (OOD) tasks to assess their robustness and generalization. As in-distribution task sets, we use the training datasets of 400 and 100 `RA` DFAs for their corresponding policies. For OOD evaluation, we consider other task sets that are not used during training: (i) 139 `R` tasks, (ii) the remaining 79 `RA` DFA-instruction pairs not used during training, and (iii) 102 `RAD` tasks. Table 1 reports the results over five random seeds, each run for 1,000 episodes. Across all evaluation settings, policies conditioned on RAD embeddings consistently outperform those conditioned solely on instruction embeddings, demonstrating strong generalization both within and beyond the training distribution.

Notably, even when trained on only 100 `RA` tasks, policies conditioned on RAD embeddings generalize effectively to unseen `R`, `RA`, and `RAD` task sets, whereas the performance of policies that solely utilize natural-language instructions degrades as the training task set size decreases. This contrast indicates that RAD embeddings encode semantically meaningful task structure with an inductive bias that supports generalization under limited supervision. Taken together, these results provide strong evidence that, when DFA representations of tasks are available, incorporating RAD embeddings into policy learning yields more robust and scalable task-conditioned controllers.

6. Discussion

We presented an empirical comparison of formal specification and language instruction embeddings in TC-RL. First, we provided a brief overview of RAD Embeddings, pretrained latent DFA representations. Second, we introduced a method for synthesizing DFA-instruction pairs by iteratively refining natural-language instructions through counterexample-guided learning with LLMs. Third, we conducted an empirical study comparing RAD Embeddings and instruction embeddings in TC-RL across multiple dimensions: latent space structure, sample efficiency, and generalization. Our results show that RAD Embeddings enable optimal task-conditioned policy learning.

Our method for generating DFA-instruction pairs has potential broader applicability beyond this comparative study. In limited domains such as digital assistants or coding agents, generated DFA-instruction datasets can be used to train neural models for auto-formalization – automatically translating natural-language instructions to formal specifications. While LLMs can also perform such auto-formalization tasks directly through prompting, combining learned neural translators with LLM-based approaches can help build trust in AI systems through redundancy.

Our results demonstrate that RAD Embeddings and instruction embeddings can be combined to improve task-conditioned policy learning, leveraging both the precision of formal specifications and the rich semantic content of natural language. However, a crucial direction for future work is

to demonstrate what additional capabilities can be gained from instruction embeddings. In partially observable environments or complex dynamics, natural language instructions might provide information about unobserved aspects of the task that cannot be captured by a DFA. Such information could enable optimal policy learning in settings where formal specifications alone are insufficient. We leave the investigation of this matter to future work, along with exploring how to best exploit the complementary strengths of both representation modalities in more challenging domains.

References

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. A framework for transforming specifications in reinforcement learning. In *Principles of Systems Design*, volume 13660 of *Lecture Notes in Computer Science*, pages 604–624. Springer, 2022.

Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. Policy synthesis and reinforcement learning for discounted LTL. In *CAV (1)*, volume 13964 of *Lecture Notes in Computer Science*, pages 415–435. Springer, 2023.

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. π_0: A vision-language-action flow model for general robot control. *CoRR*, abs/2410.24164, 2024.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*. OpenReview.net, 2022.

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. RT-1: robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

Roger Creus Castanyer, Faisal Mohamed, Pablo Samuel Castro, Cyrus Neary, and Glen Berseth. ARM-FM: automated reward machines via foundation models for compositional reinforcement learning. *CoRR*, abs/2510.14176, 2025.

Zijian Guo, Ilker Isik, H. M. Sabbir Ahmad, and Wenchao Li. One subgoal at a time: Zero-shot generalization to arbitrary linear temporal logic requirements in multi-task reinforcement learning. *CoRR*, abs/2508.01561, 2025.

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *FORMATS*, volume 12288 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2020.

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

Hengyuan Hu and Dorsa Sadigh. Language instructed reinforcement learning for human-ai coordination. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 13584–13598. PMLR, 2023.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Intell. Res.*, 73:173–208, 2022.

Mathias Jackermeier and Alessandro Abate. Deepltl: Learning to efficiently satisfy complex LTL specifications for multi-task RL. In *ICLR*. OpenReview.net, 2025.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *NeurIPS*, pages 13021–13030, 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In *NeurIPS*, pages 10026–10039, 2021.

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In *NeurIPS*, 2023.

Juntao Ren, Priya Sundaresan, Dorsa Sadigh, Sanjiban Choudhury, and Jeannette Bohg. Motion tracks: A unified representation for human-robot transfer in few-shot imitation learning. In *ICRA*, pages 8802–8810. IEEE, 2025.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *ICLR*. OpenReview.net, 2024.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A. Seshia. Ltl-constrained policy optimization with cycle experience replay. *Trans. Mach. Learn. Res.*, 2025, 2025.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415. ACM, 2006.

Sumedh Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Biyik, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. Roboclip: One demonstration is enough to learn robot policies. In *NeurIPS*, 2023.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. Ltl2action: Generalizing LTL instructions for multi-task RL. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 10497–10508. PMLR, 2021.

Marcell Vazquez-Chanlatte, Karim Elmaaroufi, Stefan J. Witwicki, Matei Zaharia, and Sanjit A. Seshia. L*lm: Learning automata from demonstrations, examples, and natural language. In *NeuS*, volume 288 of *Proceedings of Machine Learning Research*, pages 543–569. PMLR, 2025.

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

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

Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. Compositional automata embeddings for goal-conditioned reinforcement learning. In *NeurIPS*, 2024.

Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. Provably correct automata embeddings for optimal automata-conditioned reinforcement learning. In *NeuS*, volume 288 of *Proceedings of Machine Learning Research*, pages 661–675. PMLR, 2025a.

Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, Ameesh Shah, Hanna Krasowski, and Sanjit A. Seshia. Automata-conditioned cooperative multi-agent reinforcement learning. *CoRR*, abs/2511.02304, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Cambridge Yang, Michael L. Littman, and Michael Carbin. On the (in)tractability of reinforcement learning for LTL objectives. In *IJCAI*, pages 3650–3658. ijcai.org, 2022.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *CoRR*, abs/2506.05176, 2025.

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 add the `Avoid` part, call it ϕ.
For each stuttering symbol of ϕ, 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.
$\phi \leftarrow \text{minimize}(\phi)$
for $i = 1$ to m (where $m \sim$ Uniform) **do**
    $\phi' \leftarrow$ Mutate ϕ, i.e., randomly change a transition
    $\phi' \leftarrow$ Make accepting states of ϕ' sinks
    $\phi' \leftarrow \text{minimize}(\phi')$
    if ϕ' is not a trivial DFA (i.e., $\phi_\top$ or $\phi_\bot$) **then**
        $\phi \leftarrow \phi'$
    end if
end for
return ϕ

Appendix A. `ReachAvoidDerived` (`RAD`) Tasks

To generalize to a large class of tasks, we first need to identify a prior DFA distribution $\iota_\Phi \in \Delta(\Phi)$, which defines the DFA space Φ as in Equation 3, and from which we can sample for training the encoder as well as learning TC-RL policies. Recall that we want Φ 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 ι_Φ and treat Φ 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 ι_Φ 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 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 Yalcinkaya et al. (2024), 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."

We utilize the graph structure of DFAs and introduce `ReachAvoidDerived` (`RAD`) DFAs, enabling generalization to various task classes (Yalcinkaya et al. (2024)). These are randomly mutated `Reach` and `ReachAvoid` DFAs, defining a richer structure than all the task classes discussed previously. We have empirically shown that task-conditioned policies trained on `RAD` DFAs generalize to other task classes in both single- (Yalcinkaya et al. (2024)) and multi-agent settings (Yalcinkaya et al. (2025b)). Algorithm 1 presents the high-level procedure for sampling `RAD` DFAs. To train the encoder $\mathcal{E}$, we use `RAD` DFAs as our prior task generator distribution ι_Φ, which implicitly defines a DFA space Φ. Next, we continue with the details of our training procedure.

Appendix B. Automata Encoder Architecture

Here, we discuss the neural network architecture used for the DFA encoder. With an ethos similar to Section A, we utilize the graph structure of DFAs and employ a **Graph Attention Network** (GATv2), a graph neural network architecture by Brody et al. (2022), with a minor modification to include edge features in message aggregation. Below are details of our DFA encoder architecture.

Given a DFA $\phi = \langle Q, \Sigma, \delta, q_0, F \rangle$, we construct a graph $G = (V, E, h, e)$, where

- V is the nodes, containing a node for each state of ϕ,
- E is the edges, containing an edge for each transition of ϕ,
- 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:

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

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 receives messages from all n nodes and therefore encodes information about the entire DFA. We then pick the feature vector of this node, i.e., the initial state, as the embedding of the given DFA task ϕ.

Remark 4 *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* **??** *– see our previous work for more details Yalcinkaya et al. (2024).*

Appendix C. Policy Architecture and Hyperparameters

Below, we present the details of the high-level architecture in Figure 2.

- **Observation Encoder (CNN)** processes environment observations through a convolutional neural network with layers [16, 32, 64], each using a 2×2 kernel, and ReLU activation.
- **Task Encoder** depends on the configuration:
  - In the RAD Embeddings mode, the given DFA is passed through the encoder to produce its embedding.
  - In the language instruction embeddings mode, the Qwen3 Embedding of the corresponding instruction is computed.
  - Optionally, these can be combined by concatenating the embeddings.
- **Policy Head** is a multilayer perceptron with layers [64, 64, 64, $|A|$], each hidden layer using ReLU activation and orthogonal initialization. The final layer outputs logits over the action space. A categorical distribution is constructed from these logits, and actions are sampled from it.
- **Value Head** is a multilayer perceptron with layers [64, 64, 1], each hidden layer using ReLU activation and orthogonal initialization. It outputs a scalar state-value estimate.

We train using PPO by Schulman et al. (2017) – the hyperparameters are given in Table 2.

Hyperparameter	**Value**
Learning rate	3×10^{-4}
Number of environments	16
Number of steps per rollout	128
Total timesteps	1,000,000
Update epochs	10
Number of minibatches	8
Discount factor (γ)	0.99
GAE λ	0.95
Clipping coefficient	0.2
Entropy coefficient	0.01
Value function coefficient	0.5
Max gradient norm	0.5

Table 2: PPO hyperparameters used in experiments.