PLATOLTL: SCALING LTL-GUIDED MULTI-TASK RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Linear temporal logic (LTL) has emerged as a powerful formalism for specifying structured, temporally extended tasks in multi-task reinforcement learning (RL). However, while existing approaches in LTL-guided multi-task RL demonstrate success in simple environments, they suffer from challenges in *representation learning* and *exploration efficiency* when applied to high-dimensional environments with *parameterized* specifications. We present PlatoLTL, which elevates state-of-the-art methods to address both challenges. We model atomic propositions as instances of *atomic predicates* and inject task parameters directly into the goal embedding to enable efficient generalization. We also leverage *simple priors* on closeness to predicate satisfaction to enable and accelerate learning of complex tasks without biasing the optimal policy. We validate our approach on challenging environments including robotic manipulation and multi-drone navigation.

1 INTRODUCTION

A key challenge in robotics and artificial intelligence (AI) is to create agents capable of performing complex, long-horizon tasks. A rapidly growing *neuro-symbolic* paradigm uses formal languages to define task objectives and guide the learning of policies in frameworks such as reinforcement learning (RL). Accordingly, linear temporal logic (LTL; Pnueli, 1977) has recently been adopted as a powerful formalism for specifying complex, temporally extended, and non-Markovian tasks in RL (Hasanbeig et al., 2023) — a paradigm particularly valuable in settings that demand precise, verifiable behavior yet afford no large pre-training datasets, as is typical of safety-critical applications (Baier & Katoen, 2008). Recent approaches, which we refer to as *backbones*, learn *multi-task* policies that zero-shot generalize across tasks, i.e., LTL specifications (Jackermeier & Abate, 2025).

However, when scaling to high-dimensional environments such as robotics, backbones face two key challenges. The first is *representation learning*. LTL tasks are built from *atomic propositions*, i.e., Boolean facts about the environment. For these environments, propositions may be *parameterized*: a robotic arm may need to pick up an object obj, move it to position $\mathbf{x}$, or rotate its gripper to orientation $\boldsymbol{\theta}$. Backbones must either discretize the parameter space, which scales poorly with resolution and dimension, or append parameters to observations, forcing the policy to learn complex associations between goal and state embeddings, which degrades in multi-task settings. We instead model propositions as instances of *atomic predicates* and inject parameters into the goal embedding; this *inductive bias* yields low-dimensional per-predicate embeddings that generalize across parameters.

The second challenge is *exploration efficiency*; a backbone's reward signal is typically *sparse* and does not provide guidance on *where* the next subgoal is, instead only signaling satisfaction or violation of the full task. For high-dimensional environments such as robotic manipulation, learning subgoals such as pick-and-place becomes infeasible under unguided exploration. We introduce *robustness metrics* representing *simple priors* about closeness to predicate satisfaction, and compose these metrics into a shaped reward that guides exploration without biasing the optimal policy.

Our contributions are: (i) We present the *Plato embedding module*, which greatly improves learning for *parameterized* LTL specifications by injecting predicate parameters into the goal embedding. (ii) We present *Plato potential-based reward shaping* (P^2BRS), which enables straightforward injection of *coarse* domain knowledge to guide learning of complex tasks while keeping the optimal policy unbiased. (iii) We introduce high-dimensional robotics environments for LTL-guided RL including manipulation as well as quadruped and multi-drone navigation. (iv) We show that PlatoLTL elevates state-of-the-art backbones, raising near-zero success rates to 75–97% in our hardest environment.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

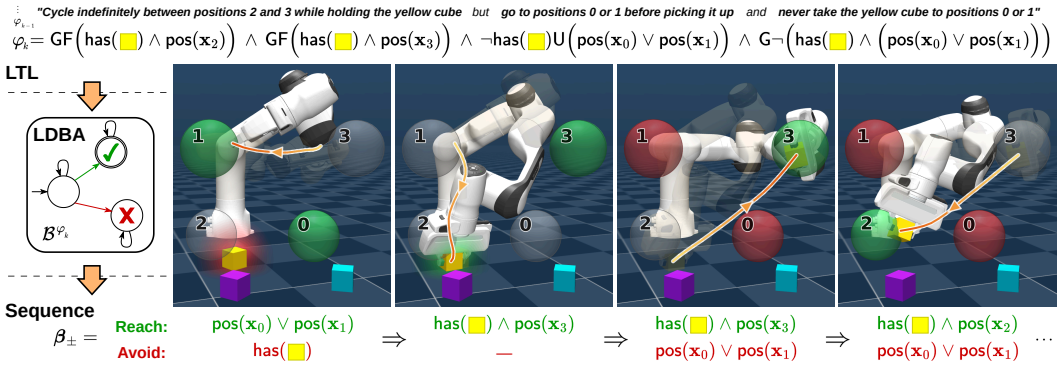


Figure 1: During training, we convert LTL specifications into LDBAs and use depth-first search to extract *reach-avoid sequences* corresponding to LTL satisfaction. P^{2}BRS combines these sequences with per-predicate *robustness metrics* to construct dense potential functions to guide policy learning.

2 Related Work

LTL-Guided RL. Methods for learning LTL-guided policies generally follow one of two approaches. The first converts the LTL formula into an automaton and conditions the policy directly on the automaton's states or transitions (Icarte et al., 2018; Hahn et al., 2019; Hasanbeig et al., 2020; 2023; Bozkurt et al., 2020; Voloshin et al., 2023); since these correspond only to a particular specification, such methods produce *single-task* policies. In contrast, the methods we refer to as *backbones* condition the policy on a learned task *representation*, i.e., a goal, in a space shared across specifications (Vaezipoor et al., 2021; Qiu et al., 2023; Jackermeier & Abate, 2025; Guo et al., 2025; Jackermeier et al., 2026), yielding *multi-task* policies when trained over a distribution of specifications. Backbones typically compose goal embeddings from a fixed, discrete vocabulary of propositions, which scales poorly to the *parameterized* specifications natural in robotics: binning continuous parameters into propositions is polynomial in resolution and exponential in dimension, while appending parameters to the observation (Jothimurugan et al., 2021; Shukla et al., 2024), even with feature-wise modulation (Hatanaka et al., 2025), forces the policy to learn complex associations between goal and state embeddings that degrade in multi-task settings. We instead model propositions as instances of *atomic predicates* and inject parameters into the goal embedding.

Exploration in LTL-Guided RL. Since LTL rewards are typically *sparse*, exploration efficiency is a key challenge. Training curricula (Jackermeier & Abate, 2025) and hindsight relabeling (Qiu et al., 2023) can speed up learning, but provide no explicit guidance towards states satisfying complex subgoals, which may be visited impractically rarely. Potential functions over automaton states (Camacho et al., 2019; Icarte et al., 2022; Bagatella et al., 2025) preserve the optimal policy, but are constant in the environment state, and thus provide no gradient towards satisfying the current subgoal. Ground-Compose-Reinforce (Li et al., 2025) composes proposition-level value functions into dense potentials, but learns them offline from a dataset of labeled trajectories, trains single-task policies, and ignores *implicit* orderings within conjunctions. Planner-derived dense rewards (Cai et al., 2023) are geometric and single-task. Alternative specification languages with *quantitative semantics* (Maler & Ničković, 2004; Li et al., 2017; Jothimurugan et al., 2019) densify the satisfaction signal, but require accurate per-predicate robustness metrics, bias optimization towards expected robustness rather than satisfaction probability, and are semantically incompatible with LTL-based backbones. In contrast, our robustness metrics are *simple priors* encoding coarse domain knowledge that require no pre-training data, we provide dense guidance in multi-task settings without biasing the optimal policy, and our tasks remain fundamentally LTL. See Appendix A for extended related work.

3 Background

Reinforcement Learning. We consider a model-free reinforcement learning (RL) setup where an agent interacts with an unknown environment modeled as a *Markov Decision Process* (MDP; Sutton & Barto, 2018). An MDP is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, \mu, r, \gamma \rangle$, with state space $\mathcal{S}$ and action space $\mathcal{A}$, where $p(s' \mid s, a) : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ and $\mu(s) \in \Delta(\mathcal{S})$ are the (un-

known) state-transition and initial state distributions, respectively, $r(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. We aim to find a stochastic policy $\pi(a \mid s) : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the *expected discounted return* $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\tau = (s_0, a_0, r_0, s_1, \dots)$ is a trajectory generated under π and p starting from $s_0 \sim \mu$, such that $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$. The *value function* of π, $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$, is the expected discounted return under π starting from s.

Linear Temporal Logic. *Linear temporal logic* (LTL; Pnueli, 1977) is a human-readable formalism for specifying properties over infinite trajectories, expressing complex temporally extended behaviors including safety, liveness, recurrence, and persistence (Manna & Pnueli, 1990). For example, Figure 1 specifies a robotic manipulation task in which a robot must cycle a cube indefinitely between two positions while never carrying it to two other positions. LTL formulae are composed from a finite set of *atomic propositions*, AP, which represent Boolean statements about the environment. The syntax of LTL is recursively defined as $\varphi ::= \top \mid \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\psi$, where $\top$ denotes true, $\mathsf{p} \in AP$ is a proposition, $\neg$ (negation) and $\wedge$ (conjunction) are standard Boolean operators, φ and ψ are LTL formulae, and X (next) and U (until) are temporal operators. We define temporal operators F (eventually) and G (always) as $\mathsf{F}\,\varphi \equiv \top\,\mathsf{U}\,\varphi$ and $\mathsf{G}\,\varphi \equiv \neg\,\mathsf{F}\,\neg\varphi$, respectively.

Let $\Sigma \subseteq 2^{AP}$ be the *alphabet*, in other words, the set of all permitted combinations of proposition truth values. Let $\sigma \in \Sigma$ be an *assignment*, i.e., the set of propositions that are true at a given time. An *ω-word* $w = (\sigma_0, \sigma_1, \sigma_2, \dots)$ is an infinite sequence of assignments; we denote the set of all ω-words over Σ as Σ^ω, such that $w \in \Sigma^\omega$. The satisfaction semantics of LTL are defined via the recursive satisfaction relation $w \models \varphi$; see Appendix B for details. To ground LTL satisfaction in an MDP, we assume access to a *labeling function* $L(s) : \mathcal{S} \to \Sigma$, which returns the set of propositions that are true for a given state s. A trajectory τ is mapped to an ω-word via its trace $\mathrm{tr}(\tau) = \big(L(s_0), L(s_1), \dots\big)$, and $\tau \models \varphi$ is shorthand for $\mathrm{tr}(\tau) \models \varphi$. We thus define the *satisfaction probability* of an LTL formula φ under policy π in MDP $\mathcal{M}$ as $\Pr(\pi \models \varphi) = \mathbb{E}_{\tau\sim\pi}\left[\mathbb{1}[\tau \models \varphi]\right]$.

Büchi Automata. A more convenient way of reasoning about LTL satisfaction is to use *Büchi automata* (Büchi, 1966), which are finite-state machines that can be employed to monitor the progress towards satisfying a given LTL specification. In this work we consider *limit-deterministic Büchi automata* (LDBAs; Sickert et al., 2016), which limit non-determinism to special transitions, called *ε-transitions*, and as such are particularly suitable for use alongside MDPs. An LDBA is defined as a tuple $\mathcal{B} = \langle \mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E}\rangle$, where $\mathcal{Q}$ is the finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma \subseteq 2^{AP}$ is the finite alphabet, $\delta(q, \sigma) : \mathcal{Q} \times (\Sigma \cup \mathcal{E}) \to \mathcal{Q}$ is the transition function, $\mathcal{F}$ is the set of accepting states, and $\mathcal{E}$ is the set of ε-transitions. An ω-word w is *accepted* by $\mathcal{B}$ if it induces a *run* $\mathbf{q} = (q_0, q_1, q_2, \dots)$ that visits accepting states $q \in \mathcal{F}$ infinitely often. While *almost* deterministic, LDBAs retain the expressiveness of non-deterministic Büchi automata, and they can thus be used to represent any LTL specification: given an LTL formula φ, one can automatically construct an LDBA $\mathcal{B}^\varphi = \langle \mathcal{Q}^\varphi, q_0^\varphi, \Sigma^\varphi, \delta^\varphi, \mathcal{F}^\varphi, \mathcal{E}^\varphi\rangle$ such that $\mathcal{B}^\varphi$ accepts exactly the ω-words w for which $w \models \varphi$ (Sickert et al., 2016). See Appendix C for further details on LDBAs alongside an example.

Product MDPs. A *product MDP* $\mathcal{M}^\varphi$ extends the state space of $\mathcal{M}$ to monitor the state of $\mathcal{B}^\varphi$, yielding a product state $(s, q^\varphi) \in \mathcal{S} \times \mathcal{Q}^\varphi$; see Appendix D for a formal definition. Since q^φ encodes the memory necessary to satisfy φ, this allows policies conditioned on (s, q^φ) to be *memoryless* (Hahn et al., 2019; Bozkurt et al., 2020). In practice, we never explicitly construct the product MDP, but instead update q^φ using the assignment $\sigma^\varphi = L^\varphi(s)$ observed at each time step.

4 Problem Setting

We use RL to obtain a policy $\pi(a \mid s, q^\varphi)$ that maximizes the probability of satisfying an LTL formula $\varphi \sim \xi$, where ξ is an arbitrary distribution over LTL formulae with support $\mathrm{supp}(\xi)$. The optimal policy π^* over the product MDP $\mathcal{M}^\varphi$ is thus defined as

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\varphi\sim\xi,\\ \tau\sim\pi\mid\varphi}} \left[\mathbb{1}[\tau \models \varphi]\right]. \tag{1}$$

There exist a variety of approaches to solving Equation 1; we refer to such approaches as *backbones*, which design and optimize reward functions $r^\varphi(s, q^\varphi, a, s', q'^\varphi)$ aligned with LTL satisfaction.

However, in a departure from existing literature, we model all propositions as instances drawn from a shared set of known *atomic predicates*, $Pred$, which represent *parameterized* Boolean statements about the environment, such that the truth value depends on the input parameter values. In other words, each proposition in a specification is equivalently a *predicate instance* with a particular set of fixed parameters.[1] The number of instances of each predicate, and their parameter values, vary across formulae $\varphi \sim \xi$, giving each formula its own set of predicate instances AP^φ.

Given parameters $x_\mathsf{f} \in \mathcal{X}_\mathsf{f}$, where $\mathcal{X}_\mathsf{f} \subseteq \mathbb{R}^{n_\mathsf{f}}$ is the n_f-dimensional parameter space for f, the mapping $\mathsf{f} : \mathcal{X}_\mathsf{f} \to AP^\varphi$ instantiates predicate $\mathsf{f} \in Pred$ as a proposition $\mathsf{p} \in AP^\varphi$; thus $\mathsf{p} = \mathsf{f}(x_\mathsf{f})$ is a predicate instance of f. Each predicate f has a labeling function $L_\mathsf{f}(s, x_\mathsf{f}) : \mathcal{S} \times \mathcal{X}_\mathsf{f} \to \{0, 1\}$; given fixed parameters x_f, the labeling function for each instance becomes $L_{\mathsf{f}(x_\mathsf{f})}(s) : \mathcal{S} \to \{0, 1\}$, and these combine into the labeling function $L^\varphi(s) : \mathcal{S} \to \Sigma^\varphi$ across all predicate instances in φ, where $\Sigma^\varphi \subseteq 2^{AP^\varphi}$. Our modeling assumption is thus formalized as follows:

Assumption 4.1. Given an LTL specification $\varphi \sim \xi$, all atomic propositions $\mathsf{p} \in AP^\varphi$ are such that $\mathsf{p} = \mathsf{f}(x_\mathsf{f})$ for some atomic predicate $\mathsf{f} \in Pred$ and fixed parameters $x_\mathsf{f} \in \mathcal{X}_\mathsf{f}$. While $Pred$ and $\mathcal{X}_\mathsf{f}$ are shared across specifications, each proposition's f and x_f can vary between specifications.

Note that $\mathcal{X}_\mathsf{f}$ can be a singleton, in which case f always creates the same proposition p: thus, Assumption 4.1 is compatible with atomic propositions (e.g., from standard LTL) that are not parameterized. Parameterization induces two axes of (zero-shot) generalization: over formula *structures* and over predicate *parameters*. By *multi-task*, we refer to generalization over structures; generalizing only over parameters of a fixed formula structure remains *single-task*. Our setting requires both.

Problem Statement. While Equation 1 can *in principle* be solved under these new assumptions using existing backbones[2], the parameterized specifications induced by Assumption 4.1, and the complex high-dimensional environments to which they often apply, present significant challenges in *representation learning* and *exploration efficiency* that can prevent learning altogether in practice. Accordingly, rather than introducing an entirely new backbone, we present novel methods that augment existing backbones, enabling and accelerating learning in these settings; Section 5 addresses representation learning, and Section 6 addresses exploration efficiency.

Choice of Backbone. While PlatoLTL is not tied to a particular backbone, we use StructLTL (Jackermeier et al., 2026) as the example backbone throughout this paper; assumptions regarding backbones are made clear where needed. StructLTL first trains a policy $\pi_\beta(a \mid s, \boldsymbol{\beta}_\pm)$ conditioned on an arbitrary *reach-avoid sequence* $\boldsymbol{\beta}_\pm$ of Boolean formulae of propositions. Intuitively, $\boldsymbol{\beta}_\pm$ represents an accepting run in an LDBA, informing the policy of the subgoals required for satisfaction. To obtain a learned embedding of $\boldsymbol{\beta}_\pm$, StructLTL uses a hierarchical architecture to encode the logical structure of each formula, and applies an attention mechanism to reason about future subgoals.

At evaluation time, given a new LTL specification φ, StructLTL constructs the LDBA $\mathcal{B}^\varphi$, and uses a depth-first search (DFS) to extract a reach-avoid sequence representing an accepting run from q^φ, i.e., $\boldsymbol{\beta}^\varphi_\pm = \mathrm{DFS}_{\mathcal{B}^\varphi}(q^\varphi)$; the policy π is thus induced as $\pi(a \mid s, q^\varphi) = \pi_\beta\big(a \mid s, \mathrm{DFS}_{\mathcal{B}^\varphi}(q^\varphi)\big)$. Each Boolean formula in the sequence is associated with a transition in $\mathcal{B}^\varphi$, and succinctly represents the set of assignments that trigger that transition. Since the policy was trained on arbitrary sequences, it generalizes across LDBAs, and therefore across LTL specifications. See Appendix G for details.

5 Plato Embedding Module

A key challenge in solving Equation 1 is *representation learning*: learning an encoder for q^φ that generalizes across specifications. State-of-the-art backbones map q^φ to a useful symbolic representation which they encode into a learned goal embedding (Vaezipoor et al., 2021; Jackermeier et al., 2026). The symbolic representations are typically composed from individual propositions, and the goal embeddings are thus likewise composed from embeddings of the propositions, given a learned encoder ϕ. In summary, we assume the following about our backbones:

Assumption 5.1. The backbone learns an encoder $\phi(\mathsf{p}) : \mathbb{Z}_{\geq 0} \to \mathbb{R}^{n_\phi}$ that maps the *discrete* token ID of an atomic proposition p into an n_ϕ-dimensional embedding space.

[1] While not technically needed, Appendix E draws a connection to Plato's Theory of Forms (Plato, 1992).

[2] Although our parameterized specifications formally belong to a fragment of first-order LTL (FOLTL; Gigante, 2025), due to the parameters being fixed, the logic reduces to LTL; see Appendix F for details.

To accept a *parameterized* proposition $\mathsf{p} = \mathsf{f}(x_\mathsf{f})$ induced by Assumption 4.1, such backbones must either discretize the parameter space $\mathcal{X}_\mathsf{f}$, which scales poorly with resolution and parameter dimension, or augment the MDP state s with the parameters x_f, which requires learning a complex association between propositions in the goal embedding and parameters in the state embedding.

We propose replacing the learned mapping with the *Plato embedding module*, which learns a small MLP $\phi_\mathsf{f} : \mathbb{R}^{n_\mathsf{f}} \to \mathbb{R}^{n_\phi}$ for each predicate such that $\phi\big(\mathsf{f}(x_\mathsf{f})\big) = \phi_\mathsf{f}(x_\mathsf{f})$. This simple but effective design provides a strong *inductive bias* by injecting the parameters directly into the goal structure, helping the policy learn low-dimensional geometries for each predicate in embedding space and exploit underlying structures relating the propositions, and thus facilitating generalization across parameters. The model size of each MLP ϕ_f is very small and remains orders of magnitude below the overall policy network; thus, the total parameter cost grows linearly and mildly in $|Pred|$ (which is itself small). Predicates with singleton parameter spaces reduce to the simple learned mapping.

6 Plato Potential-Based Reward Shaping

Another key challenge in solving Equation 1 is *exploration efficiency*. For existing backbones, the reward function r^φ is typically *sparse* and does not provide guidance on *how* to achieve the next subgoal, instead only signaling satisfaction or violation of the full task (Vaezipoor et al., 2021; Jackermeier & Abate, 2025). For complex, high-dimensional tasks such as robotic manipulation, learning how to execute subgoals such as pick-and-place becomes infeasible under unguided exploration.

We propose guiding exploration using *Plato potential-based reward shaping* (P^{2}BRS), which guides the learning of subgoals while remaining *unbiased* with regard to solving Equation 1 using a given backbone. We leverage potential-based reward shaping (PBRS; Ng et al., 1999), augmenting the backbone reward with the discounted difference of a potential function $\Phi(s, q^\varphi) \colon \mathcal{S} \times \mathcal{Q}^\varphi \to \mathbb{R}$, i.e., $r^\varphi_{\text{aug}} = r^\varphi + \gamma\Phi(s', q'^\varphi) - \Phi(s, q^\varphi)$. For any bounded Φ, this is well known to leave the optimal policy unchanged (Ng et al., 1999); see Appendix H for the argument in our setting.

The remainder of this section synthesizes a suitable Φ for guiding subgoal satisfaction. Section 6.1 introduces *robustness metrics* that encode how close a proposition is to satisfaction as a real-valued signal; these metrics act as *simple priors*, requiring only coarse domain knowledge while densifying the sparse reward. Section 6.1.1 bounds and adaptively calibrates these metrics, and Section 6.1.2 composes them into robustness metrics for Boolean formulae. Section 6.2 uses these to define the potential function $\Phi_\beta(s, \boldsymbol{\beta}_\pm)$ over reach-avoid sequences $\boldsymbol{\beta}_\pm$ of such formulae, i.e., sequences of subgoals; we use Φ_β to guide the agent along accepting runs extracted from the LDBA $\mathcal{B}^\varphi$.

6.1 From Boolean Satisfaction to Dense Guidance

Inspired by specification languages with quantitative semantics (Maler & Ničković, 2004; Li et al., 2017; Jothimurugan et al., 2019), we introduce a *robustness metric* $g_\mathsf{f}(s, x_\mathsf{f}) : \mathcal{S} \times \mathcal{X}_\mathsf{f} \to \mathbb{R} \cup \{\pm\infty\}$ for each predicate f, which indicates how close $\mathsf{f}(x_\mathsf{f})$ is to being satisfied as a dense signal that is positive on satisfaction and negative on violation, i.e., $L_\mathsf{f}(s, x_\mathsf{f}) = \mathcal{H}\big(g_\mathsf{f}(s, x_\mathsf{f})\big)$. This can be ensured by applying the following construction to some metric $\hat{g}_\mathsf{f}(s, x_\mathsf{f}) : \mathcal{S} \times \mathcal{X}_\mathsf{f} \to \mathbb{R} \cup \{\pm\infty\}$:

$$g_\mathsf{f}(s, x_\mathsf{f}) = \begin{cases} \max\big\{\delta, \hat{g}_\mathsf{f}(s, x_\mathsf{f})\big\} & \text{if } L_\mathsf{f}(s, x_\mathsf{f}) = 1, \\ \min\big\{-\delta, \hat{g}_\mathsf{f}(s, x_\mathsf{f})\big\} & \text{else,} \end{cases} \qquad \text{for arbitrarily small } \delta > 0.$$

Since we bound the robustness metrics in Section 6.1.1, *any* choice of $\hat{g}_\mathsf{f}$ leaves the optimal policy unbiased. However, not all metrics are useful for *learning*, and a good metric should provide guidance on closeness to satisfaction. This guidance can be *coarse*; for example, for learning to pick up an object with a manipulator arm (with proposition $\mathsf{has}(obj)$), negated distance between its grasp center and the estimated object center can suffice. By contrast, quantitative semantics would require a metric whose sign exactly characterizes a successful grasp; since our construction takes the sign from L_f, our robustness metrics need not be highly accurate, but instead act as *simple priors* that inject some domain knowledge to help guide the agent towards satisfying a particular proposition. If there is no domain knowledge available for a predicate f, we can trivially construct a valid robustness metric $g_\mathsf{f}(s, x_\mathsf{f}) = \pm\infty$ for $L_\mathsf{f}(s, x_\mathsf{f}) = 1/0$ to unblock the use of shaping for other predicates.

6.1.1 Bounded Metrics and Adaptive Calibration

Unbiased shaping requires that $\Phi(s, q^\varphi)$ remains *bounded*. Furthermore, arbitrary robustness metrics may have vastly different scales between different predicates, rendering comparisons unsuitable. Accordingly, we use the following transformation to obtain the bounded robustness metric ρ_f:

$$\rho_\mathsf{f}(s, x_\mathsf{f}) = \frac{g_\mathsf{f}(s, x_\mathsf{f})}{\kappa_\mathsf{f}\tilde{g}_\mathsf{f} + |g_\mathsf{f}(s, x_\mathsf{f})|}, \quad \kappa_\mathsf{f}, \tilde{g}_\mathsf{f} > 0, \text{ with } \rho_\mathsf{f} = \pm 1 \text{ when } g_\mathsf{f} = \pm\infty.$$

This transformation preserves $L_\mathsf{f}(s, x_\mathsf{f}) = \mathcal{H}\big(\rho_\mathsf{f}(s, x_\mathsf{f})\big)$, while bounding the output of ρ_f to $[-1, 1]$ and tuning the sensitivity of ρ_f such that $|g_\mathsf{f}(s, x_\mathsf{f})| = \kappa_\mathsf{f}\tilde{g}_\mathsf{f} \iff |\rho_\mathsf{f}(s, x_\mathsf{f})| = 0.5$.

The *initial robustness scale* $\tilde{g}_\mathsf{f}$ and *robustness scale factor* κ_f control the breadth of guidance from the decision boundary; a larger $\kappa_\mathsf{f}\tilde{g}_\mathsf{f}$ yields broader guidance with a gradient that remains non-flat further from the boundary, while a smaller $\kappa_\mathsf{f}\tilde{g}_\mathsf{f}$ yields sharper guidance close to the boundary but a flatter gradient when at a distance. Thus, the *robustness scale* $\kappa_\mathsf{f}\tilde{g}_\mathsf{f}$ acts as a normalizer for g_f.

The initial robustness scale acts as an initial estimate for the normalizer; we obtain $\tilde{g}_\mathsf{f}$ as the quantile q_g of $\mathcal{D}_\mathsf{f}^{n_g}$, i.e., $\tilde{g}_\mathsf{f} = Q_{q_g}\big(\mathcal{D}_\mathsf{f}^{n_g}\big)$, where $\mathcal{D}_\mathsf{f}^{n_g} = \big\{|g_\mathsf{f}[i]| \,\big|\, \delta < |g_\mathsf{f}[i]| < \infty\big\}_{i=1}^{n_g}$ consists of the first n_g data points encountered during training for the magnitude of an active g_f. Before obtaining $\tilde{g}_\mathsf{f}$, a reasonable placeholder value is $\tilde{g}_\mathsf{f} = 1$, and $\kappa_\mathsf{f} = 1$ as well during this phase.

As training progresses, the distribution of agent trajectories shifts, and thus the distribution of robustness metric values shifts. We adjust κ_f accordingly; let $\mathcal{D}_\mathsf{f}^{n_\kappa} = \big\{{}^{|g_\mathsf{f}[i]|}/_{\tilde{g}_\mathsf{f}} \,\big|\, \delta < |g_\mathsf{f}[i]| < \infty\big\}_{i=1}^{n_\kappa}$ be a sliding window of the n_κ most recent data points encountered during training for the magnitude of an active ${}^{g_\mathsf{f}}/_{\tilde{g}_\mathsf{f}}$ (after having obtained $\tilde{g}_\mathsf{f}$). The per-iteration update scheme for κ_f is given by:

$$\log\big(\kappa_\mathsf{f}^{(k)}\big) = \operatorname{clip}\Big(\log\big(\bar{\kappa}_\mathsf{f}^{(k)}\big), \log(\kappa_{\min}), \log(\kappa_{\max})\Big),$$

$$\text{where} \quad \log\big(\bar{\kappa}_\mathsf{f}^{(k)}\big) = (1-\lambda_\kappa)\log\big(\bar{\kappa}_\mathsf{f}^{(k-1)}\big) + \lambda_\kappa \log\Big(Q_{q_\kappa}\big(\mathcal{D}_\mathsf{f}^{n_\kappa}\big)\Big) \quad \text{and} \quad \log\big(\bar{\kappa}_\mathsf{f}^{(0)}\big) = 0.$$

The scale factor therefore tracks the agent's typical distance from the decision boundary: trajectories further away yield a larger κ_f and broader, more global shaping that indicates roughly where the boundary lies, while trajectories closer in yield a smaller κ_f and finer, more local shaping that resolves its exact location. The online calibration scheme thus acts as an *automatic curriculum*. Note that κ_f and $\tilde{g}_\mathsf{f}$ are frozen per episode to ensure telescoping of the potential function.

6.1.2 Dense Guidance for Boolean Formulae

Given per-proposition robustness metrics, we seek a robustness metric ρ_β for a Boolean formula β composed from those propositions. We assume that β is in disjunctive normal form (DNF), which holds for all Boolean formulae β_i^+ and β_i^- in a sequence $\boldsymbol{\beta}_\pm^\varphi$ extracted as in Section 6.2:

Assumption 6.1. $\beta = \bigvee_{k=1}^{n_\beta} \bigwedge_{j=1}^{m_{\beta k}} l_{kj}$, where all $l_{kj} \in \{\mathsf{f}(x_\mathsf{f}), \neg\mathsf{f}(x_\mathsf{f})\}$ and $\mathsf{f} \in \mathit{Pred}, x_\mathsf{f} \in \mathcal{X}_\mathsf{f}$.

A natural choice negates ρ_f for negated literals, as in quantitative semantics (Maler & Ničković, 2004), and takes the $\min$ over the literals of each clause and the $\max$ over clauses (as also in Li et al., 2025). However, the $\min$ ignores *implicit* orderings induced by the environment dynamics, where satisfying one literal first requires satisfying another and then maintaining it. For example, a manipulator tasked with $\mathsf{has}(obj) \wedge \mathsf{pos}(x, y, z)$, i.e., holding obj with its grasp center at (x, y, z), must first grasp the object and then carry it there, yet the $\min$ guides the gripper towards a point between the object and the target (Appendix N.3.2), leaving the grasp to be discovered by chance.

We resolve this with *persistence gating*. We annotate each predicate f as either *persistent* ($\mathsf{f} \in \mathit{Pred}^+ \subseteq \mathit{Pred}$) or not, where a persistent predicate is one whose satisfaction typically carries over while the agent works towards the other literals of a clause, and should thus be prioritized. In the example above, a held object travels with the gripper, so has is persistent, whereas the gripper position does not survive fetching the object, so pos is not; the same applies to, e.g., possessing an item or riding in a vehicle. Like the robustness metrics, these binary labels represent coarse domain knowledge and affect only the guidance, not the optimal policy.

Persistence gating treats each clause c_k as a two-stage subgoal: first satisfy its persistent literals, then the rest while maintaining them. Let $h_k(s)$ and $t_k(s)$ be the $\min$ of the literal robustness metrics over

the persistent and non-persistent literals of c_k, respectively. We split the unsatisfied range $[-1, 0)$ at $-\sigma$, for $\sigma \in (0, 1)$, so that the first stage occupies $[-1, -\sigma)$ and tracks only h_k, the second stage occupies $[-\sigma, 0)$ and tracks only t_k, and a satisfied clause keeps the plain $\min$ over literals:

$$\rho_{c_k}(s) = h_k(s) \wedge_\sigma t_k(s), \quad a \wedge_\sigma b = \begin{cases} -\sigma + (1-\sigma)\, a & \text{if } a < 0, \\ \sigma\, b & \text{if } a \geq 0 > b, \\ \min(a, b) & \text{otherwise,} \end{cases} \quad \rho_\beta(s) = \max_k \rho_{c_k}(s).$$

Every second-stage state thus scores above every first-stage state, so ρ_{c_k} guides the agent to satisfy persistent literals first (e.g., grasping the object before moving it). Clauses whose literals are all persistent or all non-persistent use the plain $\min$ over literals; see Appendix I for the full definition.

6.2 Sequence-Conditioned Potential Function

We note that under PBRS (and thus P^{2}BRS), the potential function acts as an a priori initialization of the value estimates (Wiewiora, 2003). Thus, given an n-step reach-avoid sequence of Boolean formulae $\boldsymbol{\beta}_\pm = \big((\beta_i^+, \beta_i^-)\big)_{i=0}^{n-1}$, we use $\Phi_\beta(s, \boldsymbol{\beta}_\pm) = r_{\text{rch}} \rho_{\beta_0^+}(s) - r_{\text{avd}} \rho_{\beta_0^-}(s)$, where $r_{\text{rch}}, r_{\text{avd}} \in \mathbb{R}^+$; this design induces the prior that there is increasing value when closer to satisfying the current reach subgoal, and that there is decreasing value when closer to satisfying the current avoid subgoal.

During training, given an LTL specification φ, we apply DFS to extract a valid sequence representing an accepting run from q^φ, i.e., $\boldsymbol{\beta}_\pm^\varphi = \text{DFS}_{\mathcal{B}^\varphi}(q^\varphi)$, and compute $\Phi(s, q^\varphi) = \Phi_\beta\big(s, \text{DFS}_{\mathcal{B}^\varphi}(q^\varphi)\big)$; see Appendix J for details on extracting $\boldsymbol{\beta}_\pm^\varphi$. If the training-time policy is also sequence-conditioned, i.e., $\pi_\beta(a \mid s, \boldsymbol{\beta}_\pm)$, as is the case for StructLTL, then we forgo the DFS and use $\Phi_\beta(s, \boldsymbol{\beta}_\pm)$ directly.

Terminal Potential Function. Unbiased return assumes an infinite horizon; however, this does not hold in the case of early terminations where $T < \infty$, e.g., upon satisfaction or violation of a sequence or any environment-specific termination conditions, which yields a bias of $\gamma^T \Phi_\beta(s_T, \boldsymbol{\beta}_{\pm,T})$. To eliminate bias, we override $\Phi_\beta(s_T, \boldsymbol{\beta}_{\pm,T}) = 0$ for all terminations, regardless of episode outcome.

Robustness Metric for ε-Subgoals. An ε-transition appears at most once per sequence and only in the reach half. Such transitions represent the non-deterministic jumps required for *persistence* (FG) tasks, where the agent must satisfy some property for all time after the jump. An ε-transition has no well-defined robustness metric of its own, but it is always immediately followed by a non-empty reach subgoal, i.e., the property to be persistently satisfied, which must already hold for the jump to be taken. Since exploration for the jump is trivial, we set $\rho_{\beta_0^+}(s) = \rho_{\beta_1^+}(s)$ whenever $\beta_0^+ = \varepsilon$.

7 Experiments

We conduct experiments to answer the following research questions: (**Q1**) How does the Plato embedding module compare to existing multi-task approaches for learning *parameterized* LTL specifications? (**Q2**) Does P^{2}BRS enable and accelerate learning in *high-dimensional* environments?

Baselines. We compare the Plato embedding module ("Plato") to three baseline approaches for embedding parameterized LTL specifications. "Discretize" bins parameters into a discrete set of embeddings spanning the parameter space, representing the unaugmented backbone (Jackermeier et al., 2026). Inspired by Jothimurugan et al. (2021), "Concatenate" appends the parameters to the observation space. Finally, SIAMS (Hatanaka et al., 2025) uses Feature-wise Linear Modulation (FiLM; Perez et al., 2018) to transform the backbone's state embedding using the parameters.

We do not include the observation reduction scheme of Guo et al. (2025), which remaps observation components to generic "reach" and "avoid" observations. This remapping assumes that the observation space can be partitioned into proposition-specific observations, produced by identical observation functions under output transformation, and that a "fusion operator" can meaningfully fuse them. This assumption fails for our environments, even with parameters appended to the observations; though proposition-specific, the parameters cannot be fused into conjunctions or disjunctions.

Environments. Environments from the literature are typically simple abstractions with a small set of propositions shared by all specifications (Jackermeier et al., 2026). We instead introduce high-dimensional, predicate-based robotics environments supporting high specification complexity. This enables evaluating generalization at scales of complexity beyond existing benchmarks.

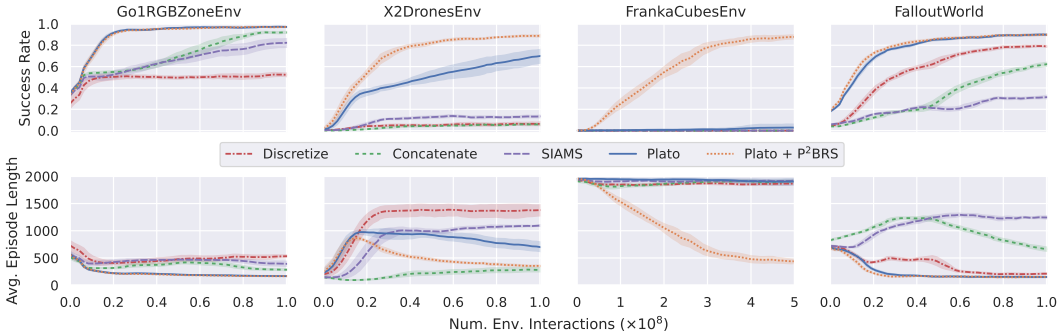


Figure 2: Evaluation curves during training for *finite*-horizon specifications.

The continuous *Go1RGBZoneEnv* environment expands upon Vaezipoor et al. (2021)'s *ZoneEnv*, and requires a Unitree Go1 quadruped robot to navigate between colored zones. To observe zones, the quadruped has a LiDAR sensor that measures the distance and RGB value of the closest zone in each bin. The policy controls the velocity command for a simple pre-trained locomotion controller.

The continuous *X2DronesEnv* environment loosely expands upon Voloshin et al. (2023)'s *FlatWorld*, and requires two Skydio X2 quadcopter drones to navigate between an arbitrary number of positions in space. The environment admits *homogeneous conjunctions* ($\mathsf{pos}_i \wedge \mathsf{pos}_j$) since the drones can be at two positions simultaneously. The policy commands each drone's model-based velocity controller.

The continuous *FrankaCubesEnv* environment loosely expands upon Jackermeier et al. (2026)'s *WarehouseEnv*, and requires a Franka Panda manipulator arm to pick and place a collection of colored cubes. The environment admits *heterogeneous conjunctions* (e.g., $\mathsf{has}_i \wedge \mathsf{pos}_j \wedge \mathsf{rot}_k$) since the arm can move to a position and orientation while holding a cube. The policy controls the grasp-center twist command, tracked by a simple model-based position controller, and the gripper state.

The discrete *FalloutWorld* environment expands upon Vaezipoor et al. (2021)'s *LetterWorld*, and requires a 2D grid-based agent to navigate between an arbitrary number of coordinates within a dense radiation field. See Appendix K.2 for details on environments, including predicates and metrics.

LTL Specifications. We evaluate the trained policy on a variety of LTL specifications representing advanced (and environment-specific) behaviors including reachability, safety, recurrence, persistence, and combinations thereof, and include both *finite*-horizon and *infinite*-horizon specifications. See Appendix K.3 for details on the template LTL specifications used for evaluation.

Evaluation Protocol. All methods are trained using the StructLTL backbone; see Appendix K.4 for training details. Tables report success rate and average episode length on success (or N/A if success rate $< 5\%$) for *finite* tasks, and average number of visits to accepting states for *infinite* tasks, over 512 episodes. Figures report success rate and average episode length over all *finite* tasks, over 16 episodes per specification. We use an evaluation horizon of 2000 steps, average results over 10 seeds, and provide 95% stratified bootstrap confidence intervals for all results (Agarwal et al., 2021).

8 Results & Discussion

Figure 2 illustrates evaluation curves during training for *finite* tasks. Table 1 presents evaluation results for *finite* tasks. See Appendix L for *infinite* tasks, including ψ_{16} illustrated by Figure 1, and see Appendix M for visualizations of example trajectories for Plato with P^{2}BRS.

Q1. Figure 2 and Table 1 show that Plato quickly achieves high success rates and low episode lengths in *Go1RGBZoneEnv* (94.9–99.7%) and *FalloutWorld* (85.5–99.9%), while the baselines generally perform substantially worse. Plato also makes steady learning progress in *X2DronesEnv*, while the baselines remain at low success rates. This is because Plato provides a strong *inductive bias* by embedding predicate parameters in the goal structure, whereas Concatenate and SIAMS must learn a complex association between propositions in the sequence embedding and parameters in the state embedding, and Discretize must learn each embedding from scratch for its hundreds to thousands of bins. Discretize is most competitive in *FalloutWorld*, where its bins coincide with grid cells.

Table 1: Evaluation results for *finite*-horizon LTL specifications.

		Success Rate (%) (↑)					Average Episode Length on Success (↓)				
	φ	Discretize	Concatenate	SIAMS	Plato	Plato + P^{2}BRS	Discretize	Concatenate	SIAMS	Plato	Plato + P^{2}BRS
Go1RGBZoneEnv	φ_1	$72.4^{+2.2}_{-2.0}$	$97.9^{+0.5}_{-0.7}$	$96.4^{+1.0}_{-1.4}$	$99.0^{+0.2}_{-0.3}$	$\mathbf{99.1}^{+0.3}_{-0.3}$	$441.4^{+28.3}_{-23.4}$	$178.9^{+11.5}_{-9.6}$	$242.5^{+32.2}_{-28.7}$	$117.9^{+4.0}_{-3.7}$	$\mathbf{116.3}^{+4.1}_{-2.2}$
	φ_2	$53.6^{+2.4}_{-3.4}$	$94.8^{+0.6}_{-0.8}$	$90.2^{+2.3}_{-3.8}$	$\mathbf{98.4}^{+0.2}_{-0.2}$	$\mathbf{98.4}^{+0.3}_{-0.2}$	$313.4^{+14.5}_{-13.1}$	$175.1^{+9.5}_{-8.0}$	$197.3^{+19.0}_{-15.6}$	$\mathbf{112.5}^{+2.7}_{-2.4}$	$114.2^{+2.6}_{-2.4}$
	φ_3	$97.9^{+0.5}_{-0.5}$	$99.2^{+0.3}_{-0.7}$	$98.8^{+0.3}_{-0.7}$	$99.0^{+0.2}_{-0.2}$	$\mathbf{99.3}^{+0.2}_{-0.2}$	$161.7^{+6.3}_{-9.3}$	$128.6^{+5.1}_{-3.9}$	$143.9^{+11.5}_{-7.7}$	$\mathbf{97.2}^{+3.2}_{-2.5}$	$101.4^{+2.3}_{-2.3}$
	φ_4	$35.9^{+1.5}_{-2.7}$	$86.1^{+2.0}_{-1.4}$	$78.3^{+3.8}_{-4.6}$	$95.6^{+0.5}_{-0.4}$	$\mathbf{96.5}^{+0.5}_{-0.6}$	$203.2^{+16.2}_{-14.1}$	$135.3^{+11.5}_{-8.1}$	$150.9^{+21.0}_{-14.5}$	$81.2^{+3.0}_{-2.3}$	$\mathbf{80.4}^{+3.6}_{-1.9}$
	φ_5	$36.1^{+2.1}_{-2.6}$	$92.7^{+0.9}_{-1.2}$	$86.8^{+1.2}_{-1.9}$	$97.3^{+0.3}_{-0.3}$	$\mathbf{97.6}^{+0.4}_{-0.4}$	$288.9^{+27.0}_{-26.6}$	$126.5^{+16.6}_{-10.9}$	$150.8^{+23.2}_{-18.7}$	$75.2^{+4.0}_{-3.6}$	$\mathbf{74.7}^{+3.3}_{-1.8}$
	φ_6	$50.1^{+2.8}_{-4.2}$	$97.2^{+0.8}_{-1.0}$	$93.3^{+1.9}_{-2.5}$	$\mathbf{99.7}^{+0.2}_{-0.1}$	$99.6^{+0.1}_{-0.1}$	$231.2^{+17.6}_{-18.8}$	$95.1^{+9.7}_{-10.0}$	$123.2^{+18.4}_{-10.5}$	$46.3^{+2.6}_{-2.1}$	$\mathbf{44.1}^{+2.4}_{-1.5}$
	φ_7	$84.1^{+1.5}_{-1.4}$	$91.9^{+1.2}_{-1.1}$	$68.9^{+10.9}_{-14.9}$	$95.0^{+0.5}_{-0.9}$	$\mathbf{95.7}^{+0.5}_{-0.5}$	$1053.8^{+59.0}_{-48.2}$	$770.0^{+24.8}_{-30.0}$	$1132.8^{+117.1}_{-121.3}$	$451.8^{+17.9}_{-12.6}$	$\mathbf{442.0}^{+12.6}_{-8.6}$
	φ_8	$3.1^{+1.1}_{-1.3}$	$83.0^{+2.8}_{-1.9}$	$64.1^{+8.1}_{-13.3}$	$94.9^{+0.4}_{-0.4}$	$\mathbf{95.5}^{+0.7}_{-1.0}$	N/A	$649.6^{+64.1}_{-42.3}$	$912.7^{+103.5}_{-97.0}$	$352.6^{+14.3}_{-13.4}$	$\mathbf{343.4}^{+16.7}_{-7.7}$
X2DronesEnv	φ_9	$26.8^{+4.1}_{-2.6}$	$22.9^{+2.1}_{-3.0}$	$34.3^{+3.5}_{-2.5}$	$91.3^{+2.7}_{-6.9}$	$\mathbf{96.2}^{+0.4}_{-0.4}$	$554.8^{+35.1}_{-31.9}$	$221.8^{+17.5}_{-21.7}$	$356.0^{+14.0}_{-16.7}$	$133.7^{+38.9}_{-24.4}$	$\mathbf{99.6}^{+5.5}_{-4.5}$
	φ_{10}	$8.2^{+3.2}_{-2.3}$	$8.8^{+1.0}_{-1.9}$	$24.4^{+2.4}_{-5.8}$	$85.2^{+3.0}_{-7.2}$	$\mathbf{89.5}^{+1.2}_{-0.7}$	$868.1^{+76.1}_{-78.9}$	$281.2^{+46.5}_{-24.4}$	$542.8^{+11.7}_{-15.4}$	$191.1^{+43.8}_{-22.1}$	$\mathbf{157.5}^{+4.9}_{-4.3}$
	φ_{11}	$0.1^{+0.2}_{-0.1}$	$0.7^{+1.0}_{-0.4}$	$1.7^{+0.7}_{-0.4}$	$60.8^{+7.2}_{-8.8}$	$\mathbf{93.0}^{+0.4}_{-0.5}$	N/A	N/A	N/A	$452.6^{+34.7}_{-30.5}$	$\mathbf{299.2}^{+18.0}_{-16.6}$
	φ_{12}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$37.5^{+9.6}_{-10.8}$	$\mathbf{85.9}^{+1.0}_{-1.2}$	N/A	N/A	N/A	$828.1^{+68.2}_{-59.3}$	$\mathbf{571.4}^{+18.5}_{-20.0}$
	φ_{13}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.1}_{-0.0}$	$0.2^{+0.2}_{-0.1}$	$45.4^{+9.3}_{-11.7}$	$\mathbf{84.4}^{+1.0}_{-1.0}$	N/A	N/A	N/A	$606.4^{+57.4}_{-44.4}$	$\mathbf{364.7}^{+10.1}_{-14.7}$
	φ_{14}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.1}_{-0.0}$	$94.3^{+1.4}_{-1.7}$	$\mathbf{95.3}^{+1.1}_{-1.0}$	N/A	N/A	N/A	$543.2^{+12.0}_{-13.8}$	$\mathbf{492.9}^{+11.7}_{-13.0}$
	φ_{15}	$10.9^{+4.0}_{-3.1}$	$16.4^{+1.2}_{-1.9}$	$37.4^{+1.7}_{-1.6}$	$\mathbf{90.6}^{+1.4}_{-1.6}$	$90.0^{+1.5}_{-1.4}$	$919.0^{+50.9}_{-50.1}$	$257.0^{+26.7}_{-19.2}$	$592.7^{+23.1}_{-27.1}$	$202.3^{+19.8}_{-14.8}$	$\mathbf{171.4}^{+7.4}_{-6.0}$
	φ_{16}	$0.1^{+0.1}_{-0.1}$	$0.7^{+0.2}_{-0.2}$	$2.1^{+0.7}_{-0.6}$	$68.5^{+6.1}_{-8.2}$	$\mathbf{90.0}^{+1.0}_{-1.2}$	N/A	N/A	N/A	$385.3^{+54.7}_{-42.0}$	$\mathbf{247.3}^{+8.5}_{-9.1}$
FrankaCubesEnv	φ_{17}	$0.0^{+0.1}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$14.4^{+15.2}_{-5.6}$	$\mathbf{97.1}^{+0.5}_{-1.4}$	N/A	N/A	N/A	$939.9^{+57.9}_{-48.5}$	$\mathbf{255.7}^{+5.5}_{-9.8}$
	φ_{18}	$0.0^{+0.1}_{-0.0}$	$0.0^{+0.1}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$3.0^{+5.9}_{-3.0}$	$\mathbf{83.2}^{+1.3}_{-1.1}$	N/A	N/A	N/A	N/A	$\mathbf{95.7}^{+7.2}_{-7.4}$
	φ_{19}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$\mathbf{85.7}^{+1.7}_{-1.7}$	N/A	N/A	N/A	N/A	$\mathbf{255.9}^{+24.6}_{-17.6}$
	φ_{20}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$\mathbf{85.0}^{+2.3}_{-2.0}$	N/A	N/A	N/A	N/A	$\mathbf{299.1}^{+23.3}_{-19.9}$
	φ_{21}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$3.2^{+7.7}_{-3.2}$	$\mathbf{86.5}^{+1.5}_{-1.0}$	N/A	N/A	N/A	N/A	$\mathbf{97.1}^{+7.5}_{-7.0}$
	φ_{22}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.1^{+0.3}_{-0.1}$	$\mathbf{75.7}^{+4.6}_{-5.8}$	N/A	N/A	N/A	N/A	$\mathbf{390.8}^{+49.7}_{-38.8}$
	φ_{23}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.1^{+0.4}_{-0.1}$	$\mathbf{79.5}^{+3.3}_{-5.3}$	N/A	N/A	N/A	N/A	$\mathbf{326.3}^{+29.9}_{-37.9}$
	φ_{24}	$0.0^{+0.1}_{-0.0}$	$0.0^{+0.1}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$2.6^{+7.3}_{-2.4}$	$\mathbf{81.9}^{+3.6}_{-6.9}$	N/A	N/A	N/A	N/A	$\mathbf{268.1}^{+47.2}_{-27.7}$
FalloutWorld	φ_{25}	$74.1^{+1.4}_{-1.1}$	$58.8^{+2.4}_{-2.2}$	$19.7^{+3.0}_{-1.7}$	$90.2^{+0.7}_{-1.8}$	$\mathbf{91.0}^{+0.5}_{-0.6}$	$80.7^{+5.3}_{-4.7}$	$288.6^{+32.0}_{-23.6}$	$594.5^{+27.6}_{-41.9}$	$\mathbf{62.4}^{+3.8}_{-3.2}$	$66.2^{+5.9}_{-2.4}$
	φ_{26}	$71.2^{+1.9}_{-1.5}$	$51.7^{+2.2}_{-2.3}$	$24.4^{+1.6}_{-1.1}$	$\mathbf{85.5}^{+0.7}_{-1.1}$	$85.4^{+0.5}_{-0.6}$	$131.9^{+9.9}_{-10.0}$	$307.8^{+31.7}_{-21.5}$	$608.9^{+25.2}_{-27.7}$	$\mathbf{81.2}^{+5.9}_{-5.4}$	$82.0^{+8.0}_{-6.4}$
	φ_{27}	$83.3^{+1.3}_{-1.2}$	$46.2^{+3.3}_{-3.0}$	$24.1^{+2.6}_{-2.3}$	$\mathbf{91.5}^{+1.2}_{-1.6}$	$90.8^{+1.9}_{-2.6}$	$103.3^{+7.3}_{-8.6}$	$392.5^{+29.4}_{-20.8}$	$564.7^{+25.4}_{-26.7}$	$\mathbf{75.8}^{+8.5}_{-7.7}$	$79.7^{+1.8}_{-10.6}$
	φ_{28}	$82.3^{+1.0}_{-0.8}$	$63.2^{+2.2}_{-2.3}$	$32.9^{+2.5}_{-2.1}$	$91.8^{+0.4}_{-0.6}$	$\mathbf{92.3}^{+0.5}_{-0.6}$	$92.7^{+7.2}_{-4.9}$	$249.6^{+24.2}_{-15.6}$	$492.7^{+21.3}_{-18.9}$	$\mathbf{57.0}^{+3.2}_{-2.4}$	$59.8^{+3.8}_{-2.7}$
	φ_{29}	$76.8^{+1.0}_{-1.0}$	$48.1^{+2.2}_{-1.9}$	$24.0^{+2.1}_{-1.7}$	$\mathbf{89.7}^{+1.0}_{-0.7}$	$89.0^{+0.6}_{-0.6}$	$121.1^{+6.1}_{-8.1}$	$432.8^{+21.0}_{-26.2}$	$783.5^{+14.4}_{-21.4}$	$\mathbf{79.6}^{+2.8}_{-2.2}$	$81.8^{+8.6}_{-4.6}$
	φ_{30}	$\mathbf{99.6}^{+0.1}_{-0.3}$	$98.5^{+0.4}_{-0.6}$	$70.1^{+7.3}_{-7.4}$	$\mathbf{99.6}^{+0.1}_{-0.1}$	$99.5^{+0.2}_{-0.3}$	$40.0^{+0.7}_{-0.6}$	$94.7^{+12.3}_{-10.5}$	$371.8^{+25.6}_{-24.3}$	$\mathbf{35.9}^{+0.4}_{-0.4}$	$36.7^{+0.4}_{-0.5}$
	φ_{31}	$99.7^{+0.1}_{-0.2}$	$73.5^{+6.2}_{-9.5}$	$1.0^{+0.8}_{-0.5}$	$\mathbf{99.9}^{+0.1}_{-0.1}$	$99.8^{+0.1}_{-0.1}$	$116.0^{+2.4}_{-2.5}$	$611.4^{+54.1}_{-43.7}$	N/A	$\mathbf{106.0}^{+1.2}_{-1.0}$	$108.0^{+1.1}_{-1.2}$
	φ_{32}	$78.8^{+1.2}_{-1.4}$	$63.9^{+1.3}_{-1.8}$	$43.5^{+1.7}_{-1.4}$	$\mathbf{92.5}^{+0.9}_{-1.0}$	$91.6^{+0.4}_{-0.4}$	$92.6^{+10.2}_{-8.5}$	$255.6^{+20.2}_{-14.3}$	$493.7^{+24.6}_{-24.2}$	$\mathbf{63.5}^{+3.8}_{-5.0}$	$64.0^{+3.1}_{-3.2}$

Q2. We see that P^{2}BRS markedly accelerates learning in *X2DronesEnv*, with success rates converging to 84.4–96.2% while unguided Plato is still far from convergence. The impact of P^{2}BRS is even starker for *FrankaCubesEnv*, taking Plato's success rates from near zero to 75.7–97.1%. This is to be expected, since the learning signal is extremely sparse in these high-dimensional robotics environments, especially when learning conjunctions. In *X2DronesEnv*, satisfying $\mathsf{pos}_i \wedge \mathsf{pos}_j$ requires each drone to enter specific regions in a large 3D space, which is an exceedingly rare occurrence. In *FrankaCubesEnv*, satisfying $\mathsf{has}_i \wedge \mathsf{pos}_j \wedge \mathsf{rot}_k$ requires the arm to pick up the correct cube and bring it to a specific position and orientation, which is virtually impossible under unguided exploration. Thus, a dense guiding signal is essential for effective exploration in these environments. Meanwhile, we see little to no benefit in *Go1RGBZoneEnv* and *FalloutWorld*; this suggests that exploration is not the learning bottleneck for these simpler environments with the StructLTL backbone.

Ablations. Appendix N.1 uses principal component analysis to demonstrate that Plato learns low-dimensional predicate embedding geometries. Appendix N.2 applies PlatoLTL to LTL2Action, a second backbone. Appendix N.3 ablates P^{2}BRS, validating the need for adaptive robustness scale calibration and persistence gating, and demonstrating that automaton-based PBRS is insufficient.

Limitations. The benefits of PlatoLTL scale with the task structure its inputs expose. The Plato embedding module relies on propositions being represented as instances of predicates known a priori; while such a parameterization always exists, its benefit requires the labeling function to vary structurally with the parameters, and in the absence of such structure we would not expect gains over the unaugmented backbone. Similarly, the guidance of P^{2}BRS is only as informative as the supplied robustness metrics; a valid metric always exists and shaping never biases the optimal policy, but informative metrics may be challenging to hand-specify for semantically complex predicates. Such predicates compress into a single atom the structure that a more granular specification would expose with readily specified metrics. Under weak or trivial metrics, or overly coarse specifications, P^{2}BRS reverts to unguided exploration for the affected predicates. See Appendix O for further discussion.

Conclusion. The Plato embedding module enhances representation learning for parameterized specifications, while P^{2}BRS accelerates exploration in high-dimensional robotics environments, lifting success rates from near zero to 75.7–97.1% in *FrankaCubesEnv*. Future work could investigate using uncertain and learned labeling functions in a step towards enabling real-world robotic deployment, and pre-training robustness metrics from demonstrations; see Appendix P for further discussion.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing methods. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for or assist in writing proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or interpret results. Generating synthetic datasets, translation, cleaning or reformatting datasets, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit software code (including plotting, evaluation, and tooling scripts), and to check for typos and suggest rewording to improve readability. We have reviewed all AI-assisted work. AI-assisted code was checked by the authors and tested with unit tests, and suggested rewordings were reviewed and refined by the authors before adoption. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure the reproducibility of this work. Source code for PlatoLTL, including implementations of all four environments, the pre-training pipeline for the Go1 locomotion controller, all baselines, the training and evaluation pipelines, and all ablation studies, is available via an anonymous repository linked here: https://anonymous.4open.science/r/plato-ltl-iclr-2027-anonymous-FC6E. The problem setting and its assumptions are stated in Section 4, and the Plato embedding module and P^2BRS are specified in full in Sections 5 and 6, respectively. Appendix H proves that P^2BRS leaves the optimal policy unchanged, and Appendix F justifies the equivalence of our parameterized specifications to LTL. Appendices G and J detail the StructLTL backbone and the reach-avoid sequence extraction procedure used by both the backbone and P^2BRS, and Appendix N.2 describes how PlatoLTL is applied to LTL2Action. Appendix K.1 lists the software used in our implementation and the hardware used for running experiments. Appendix K.2 fully specifies each environment, including its dynamics, observation and action spaces, low-level controllers (and the pre-training of the Go1 locomotion controller), reset distributions, predicates, labeling functions, robustness metrics, and persistence annotations. The template LTL specifications used for evaluation are listed in Table 4 (Appendix K.3). Network architectures for the Plato embedding module, all baselines, and both backbones are given in Appendix K.4.1, and all PPO and P^2BRS hyperparameters are listed in Table 2. The training curriculum is detailed in Appendix K.4.3 and Table 5, and the configurations for each ablation are described in Appendix N. Finally, Section 7 describes the evaluation protocol, including the number of seeds, evaluation episodes, and evaluation horizon, as well as the construction of confidence intervals.

REFERENCES

Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Kretínský, Maximilian Prokop, and Christoph Weinhuber. Semantically Labelled Automata for Multi-Task Reinforcement Learning with LTL Instructions. In *35th International Joint Conference on Artificial Intelligence and 29th European Conference on Artificial Intelligence (IJCAI/ECAI)*, 2026.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

Masataro Asai. Unsupervised Grounding of Plannable First-Order Logic Representation from Images. In *29th International Conference on Automated Planning and Scheduling (ICAPS)*, 2019.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450*, 2016.

Marco Bagatella, Andreas Krause, and Georg Martius. Directed Exploration in Reinforcement Learning from Linear Temporal Logic. *Transactions on Machine Learning Research (TMLR)*, 2025.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert 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, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Julius R. Büchi. Symposium on Decision Problems: On a Decision Method in Restricted Second Order Arithmetic. In *Logic, Methodology and Philosophy of Science*, volume 44, pp. 1–11, 1966.

Mingyu Cai, Mohammadhosein Hasanbeig, Shaoping Xiao, Alessandro Abate, and Zhen Kan. Modular Deep Reinforcement Learning for Continuous Motion Planning With Temporal Logic. *IEEE Robotics and Automation Letters (RA-L)*, 6(4):7973–7980, 2021.

Mingyu Cai, Erfan Aasi, Calin Belta, and Cristian-Ioan Vasile. Overcoming Exploration: Deep Reinforcement Learning for Continuous Control in Cluttered Environments From Temporal Logic Specifications. *IEEE Robotics and Automation Letters (RA-L)*, 8(4):2158–2165, 2023.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives. In *35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Alexandre Donzé and Oded Maler. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2010.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-Language Models as Success Detectors. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023.

Ben Eysenbach, Sergey Levine, and Ruslan Salakhutdinov. Replacing Rewards with Examples: Example-Based Policy Search via Recursive Classification. In *35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Zining Fan and He Zhu. Imitation Learning with Temporal Logic Constraints. In *39th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition. In *32nd Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Francesco Fuggitti and Tathagata Chakraborti. NL2LTL – a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas. In *37th AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

Nicola Gigante. An Introduction to First-Order Linear Temporal Logic. In *32nd International Symposium on Temporal Representation and Reasoning (TIME)*, 2025.

Zijian Guo, İlker Işık, 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. In *39th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Vedant Gupta, Haotian Fu, Calvin Luo, Yiding Jiang, and George Konidaris. Learning Parameterized Skills from Demonstrations. In *39th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2019.

Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael J. Wooldridge. Multi-Agent Reinforcement Learning with Temporal Logic Specifications. In *20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

Shaohang Han, Joris Verhagen, and Jana Tumova. Exact Smooth Reformulations for Trajectory Optimization Under Signal Temporal Logic Specifications. In *European Control Conference (ECC)*, 2026.

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep Reinforcement Learning with Temporal Logics. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2020.

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Certified reinforcement learning with logic guidance. *Artificial Intelligence*, 322:103949, 2023.

Wataru Hatanaka, Ryota Yamashina, and Takamitsu Matsubara. Reinforcement Learning of Action and Query Policies With LTL Instructions Under Uncertain Event Detector. *IEEE Robotics and Automation Letters (RA-L)*, 8(11):7010–7017, 2023.

Wataru Hatanaka, Ryota Yamashina, and Takamitsu Matsubara. Reinforcement Learning of Flexible Policies for Symbolic Instructions With Adjustable Mapping Specifications. *IEEE Robotics and Automation Letters (RA-L)*, 10(3):2614–2621, 2025.

Matthew J. Hausknecht and Peter Stone. Deep Reinforcement Learning in Parameterized Action Space. In *4th International Conference on Learning Representations (ICLR)*, 2016.

Yanhao He and Steven Liu. Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design. In *9th International Conference on Control, Mechatronics and Automation (ICCMA)*, 2021.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *35th International Conference on Machine Learning (ICML)*, 2018.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *Journal of Artificial Intelligence Research (JAIR)*, 73:173–208, 2022.

Mathias Jackermeier and Alessandro Abate. DeepLTL: Learning to Efficiently Satisfy Complex LTL Specifications for Multi-Task RL. In *13th International Conference on Learning Representations (ICLR)*, 2025.

Mathias Jackermeier, Mattia Giuri, Jacques Cloete, and Alessandro Abate. Zero-Shot Instruction Following in RL via Structured LTL Representations. *arXiv:2602.14344*, 2026.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A Composable Specification Language for Reinforcement Learning Tasks. In *33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional Reinforcement Learning from Logical Specifications. In *35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. In *Conference on Robot Learning (CoRL)*, 2024.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to Your Favourite Deterministic Automaton. In *Computer Aided Verification (CAV)*, 2018.

Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. MegaPose: 6D Pose Estimation of Novel Objects via Render & Compare. In *Conference on Robot Learning (CoRL)*, 2022.

Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *49th IEEE Conference on Decision and Control (CDC)*, 2010.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. In a Nutshell, the Human Asked for This: Latent Goals for Following Temporal Specifications. In *10th International Conference on Learning Representations (ICLR)*, 2022.

Andrew C. Li, Zizhao Chen, Toryn Q. Klassen, Pashootan Vaezipoor, Rodrigo Toro Icarte, and Sheila A. McIlraith. Reward Machines for Deep RL in Noisy and Uncertain Environments. In *38th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Andrew C. Li, Toryn Q. Klassen, Andrew Wang, Parand A. Alamdari, and Sheila A. McIlraith. Ground-Compose-Reinforce: Grounding Language in Agentic Behaviours using Limited Data. In *39th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-Image Representations and Rewards for Robotic Control. In *40th International Conference on Machine Learning (ICML)*, 2023a.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards Universal Visual Reward and Representation via Value-Implicit Pre-Training. In *11th International Conference on Learning Representations (ICLR)*, 2023b.

Oded Maler and Dejan Ničković. Monitoring Temporal Properties of Continuous Signals. In *Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, 2004.

Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1990.

Edward J. McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35 (6):1417–1444, 1956.

Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *16th International Conference on Machine Learning (ICML)*, 1999.

Pierriccardo Olivieri, Fausto Lasca, Alessandro Gianola, and Matteo Papini. Do It for HER: First-Order Temporal Logic Reward Specification in Reinforcement Learning. In *40th AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

Jing-Cheng Pang, Xin-Yu Yang, Si-Hang Yang, Xiong-Hui Chen, and Yang Yu. Natural Language Instruction-following with Task-related Language Development and Translation. In *37th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Plato. *Republic*. Hackett Publishing, 1992.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS)*, 1977.

Ofir Press, Noah A. Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *10th International Conference on Learning Representations (ICLR)*, 2022.

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. In *37th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Willard V. Quine. The Problem of Simplifying Truth Functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv: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. *Transactions on Machine Learning Research (TMLR)*, 2025.

Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

Yash Shukla, Tanushree Burman, Abhishek N. Kulkarni, Robert Wright, Alvaro Velasquez, and Jivko Sinapov. Logical Specifications-guided Dynamic Task Sampling for Reinforcement Learning Agents. In *34th International Conference on Automated Planning and Scheduling (ICAPS)*, 2024.

Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *Computer Aided Verification (CAV)*, 2016.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *38th International Conference on Machine Learning (ICML)*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual Discounting Temporal Logic Counterfactual Experience Replay. In *40th International Conference on Machine Learning (ICML)*, 2023.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D Pose Estimation and Tracking of Novel Objects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Eric Wiewiora. Potential-Based Shaping and Q-Value Initialization are Equivalent. *Journal of Artificial Intelligence Research (JAIR)*, 19:205–208, 2003.

Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In *Robotics: Science and Systems (RSS)*, 2018.

Duo Xu and Faramarz Fekri. Generalization of temporal logic tasks via future dependent options. *Machine Learning*, 113(10):7509–7540, 2024.

Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. Compositional Automata Embeddings for Goal-Conditioned Reinforcement Learning. In *38th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Modular Deep Reinforcement Learning with Temporal Logic Specifications. *arXiv:1909.11591*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo, 2022. URL http://github.com/google-deepmind/mujoco_menagerie.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground. *arXiv:2502.08844*, 2025.

Hao Zhang, Hao Wang, and Zhen Kan. Exploiting Transformer in Sparse Reward Reinforcement Learning for Interpretable Temporal Logic Motion Planning. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):4831–4838, 2023.

Hao Zhang, Hao Wang, Xiucai Huang, Wenrui Chen, and Zhen Kan. Exploiting Hybrid Policy in Reinforcement Learning for Interpretable Temporal Logic Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the Continuity of Rotation Representations in Neural Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong T. Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Conference on Robot Learning (CoRL)*, 2023.

A Extended Related Work

LTL-Guided RL. Methods for learning LTL-guided policies can generally be classified into two approaches. The first approach converts the LTL formula into an automaton and learns a policy conditioned directly on the automaton's state or transitions (Icarte et al., 2018; 2022; Camacho et al., 2019; Yuan et al., 2019; Hahn et al., 2019; Bozkurt et al., 2020; Hasanbeig et al., 2020; 2023; Cai et al., 2021; Voloshin et al., 2023; Shah et al., 2025; Fan & Zhu, 2025; Li et al., 2025), either as a single policy or a modular set of sub-policies. However, such methods generally produce *single-task* policies, since the automaton states and transitions correspond only to a particular specification. In contrast, the methods we refer to as *backbones* condition the policy on a learned representation of the task as a goal, with the goal space shared across specifications (Vaezipoor et al., 2021; León et al., 2022; Qiu et al., 2023; Zhang et al., 2023; 2024; Yalcinkaya et al., 2024; Xu & Fekri, 2024; Jackermeier & Abate, 2025; Guo et al., 2025; Abate et al., 2026; Jackermeier et al., 2026). By training over a sufficiently wide distribution of such representations, one obtains *multi-task* policies.

For example, LTL2Action (Vaezipoor et al., 2021) encodes the syntax tree of the task LTL specification using a graph neural network (GNN; Zhou et al., 2020), updating the specification via LTL progression as the agent progresses through the task. Meanwhile, DeepLTL (Jackermeier & Abate, 2025) extracts a reach-avoid sequence of sets of assignments for LTL satisfaction from its corresponding automaton, and uses a recurrent neural network (RNN; Sherstinsky, 2020) to embed the sequence, updating it as the agent satisfies each subgoal. StructLTL (Jackermeier et al., 2026) improves upon this by representing sets of assignments as Boolean formulae, and learns *structured* task representations to better generalize across particularly complex specifications.

Learning Representations for Parameterized Specifications. Backbones typically compose goal embeddings from a fixed, discrete vocabulary of propositions known a priori. This presents a challenge when faced with *parameterized* specifications as is commonplace in high-dimensional environments such as robotics, since naïvely binning continuous parameters into discrete propositions scales poorly with resolution and parameter dimensions. The other naïve solution is to append the parameters to the observation space; such an approach works well for single-task policies, i.e., those generalizing only over the parameters of a fixed formula structure (Jothimurugan et al., 2019; 2021; Shukla et al., 2024), but degrades in multi-task settings, since the policy must learn a complex association between propositions in the goal embedding and parameters in the state embedding. Hatanaka et al. (2025) go a step further and use Feature-wise Linear Modulation (FiLM; Perez et al., 2018) to transform the backbone's state embedding using the parameters, but the method suffers the same fundamental scaling issues once more than one proposition may be relevant at a given time. In comparison, our work models propositions as parameterized instances of *atomic predicates* and injects the parameters into the goal embedding, which yields low-dimensional per-predicate embedding geometries and enables efficient generalization across parameters.

Guo et al. (2025) demonstrate generalization to unseen propositions (and thus new parameters) by remapping the observation space into universal "reach" and "avoid" observations. However, this assumes that the observation space can be partitioned into proposition-specific observations, produced by identical observation functions under output transformation with a shared structure amenable to fusion, which is generally not true (or requires significant environment-specific engineering) for high-dimensional robotics environments. For example, if a manipulator arm's observations include a table of the IDs and locations of all objects in the scene, it is not clear how to fuse conjunctions and disjunctions of propositions related to these objects into "reach" and "avoid" observations, even if the table can be partitioned into per-object rows. Our work makes no such assumptions.

Exploration for LTL-Guided RL. Another key challenge in LTL-guided RL is *exploration efficiency*, since the reward function is typically *sparse*. While a training curriculum (Jackermeier & Abate, 2025) or hindsight relabeling (Qiu et al., 2023) can speed up learning to some extent, they do not provide explicit guidance towards states that satisfy the current subgoal, meaning that for complex subgoals (such as conjunctions of propositions) in high-dimensional environments, visits of those states can be impractically rare.

Some methods define a potential function over automaton states to guide the learning of multi-step tasks without biasing the optimal policy (Camacho et al., 2019; Icarte et al., 2022; Bagatella et al., 2025). While effective when individual subgoals are simple enough to be discovered without guidance, automaton-level potentials are by construction constant in the environment state, so the shaped

reward differentiates progress only across (sparse) automaton transitions and provides no gradient *within* a subgoal; it thus does not address complex subgoals in high-dimensional environments.

Like our approach, Ground-Compose-Reinforce (GCR; Li et al., 2025) constructs potentials that vary with environment state for automaton-structured tasks (reward machines; Icarte et al., 2022), rewriting each transition's formula in DNF and composing per-literal progress signals via a $\min$ over literals and a $\max$ over clauses. Its progress signals are value functions for reaching each proposition or its negation, learned via offline RL (alongside a labeling function) from a dataset of labeled trajectories, and combined with automaton-level values obtained via value iteration. Our work differs in several respects. First, GCR addresses a single-task setting, exploiting compositionality in the construction of rewards and potentials rather than in the policy; we instead shape the training of multi-task backbones that generalize zero-shot across specifications. Second, GCR assumes a fixed propositional vocabulary with a pair of value functions per proposition, whereas our metrics are defined per parameterized predicate. Third, GCR requires a *dataset of labeled trajectories* with adequate coverage of the state space (e.g., spanning the parameter spaces of all predicates), which must be collected for each environment; our robustness metrics are simple priors requiring no pre-training data. Fourth, a plain $\min$ over literals provides no guidance on *implicit* temporal orderings within a conjunction, and can also overestimate conjunctive values (Li et al., 2025); our *persistence gating* addresses this and proves essential for heterogeneous conjunctions (Appendix N.3.2). Finally, GCR targets reward machine returns, and its authors report best results with an undiscounted shaping term that forgoes the invariance guarantee of PBRS; P^2BRS retains this guarantee (Theorem H.1) and targets LTL satisfaction, including *persistence* ($\mathsf{F}\,\mathsf{G}$) tasks, via LDBAs. The two approaches are *complementary*, as GCR's learned value functions could serve as robustness metrics for P^2BRS (Appendix P).

Cai et al. (2023) employ a path-planning-guided dense reward scheme for single-task policies, but the planner-based guidance is geometric and does not transfer to multi-task policies. In contrast, our work provides *dense* guidance in multi-task settings without biasing the optimal policy.

Other works propose using alternative specification languages with *quantitative semantics* to densify the satisfaction signal; the most well-known is signal temporal logic (STL; Maler & Ničković, 2004; Donzé & Maler, 2010), with others including truncated LTL (TLTL; Li et al., 2017) and SPECTRL (Jothimurugan et al., 2019). Such languages require *accurate* per-predicate robustness metrics since task satisfaction is reasoned about directly from the quantitative semantics rather than Boolean labeling functions, and policy optimization often seeks to maximize the expected task robustness value, which is not equivalent to maximizing task satisfaction probability (thus biasing the policy away from it). Moreover, the difference in semantics means that these languages are incompatible with LTL-based backbones, and the bespoke RL-based methods for solving them typically learn single-task policies. Meanwhile, our robustness metrics act as *simple priors* that represent coarse domain knowledge to guide learning, and the tasks we solve remain fundamentally LTL, ensuring compatibility with (multi-task) LTL-based backbones.

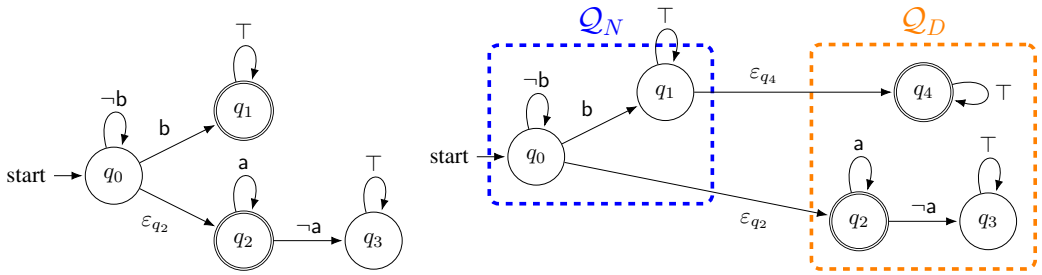
B LTL Satisfaction Semantics

Let $w[i] = \sigma_i$ and $w[i\dots] = (\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \dots)$. The satisfaction relation $w \models \varphi$ is recursively defined as follows (Baier & Katoen, 2008):

$$
\begin{aligned}
&w \models \top \\
&w \models \mathsf{p} && \iff \mathsf{p} \in w[0] \\
&w \models \neg\varphi && \iff w \not\models \varphi \\
&w \models \varphi \wedge \psi && \iff w \models \varphi \ \wedge\ w \models \psi \\
&w \models \mathsf{X}\,\varphi && \iff w[1\dots] \models \varphi \\
&w \models \varphi\,\mathsf{U}\,\psi && \iff \exists\, j \geq 0 \quad \text{s.t.} \quad w[j\dots] \models \psi \quad \wedge \quad \forall\, 0 \leq i < j : w[i\dots] \models \varphi
\end{aligned}
$$

C Limit-Deterministic Büchi Automata

We give here the formal conditions on the components of an LDBA $\mathcal{B} = \langle \mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E} \rangle$, as defined in Section 3. The state set $\mathcal{Q}$ is partitioned into two subsets $\mathcal{Q}_N$ (initial component) and $\mathcal{Q}_D$



(a) Contracted LDBA (b) Full LDBA ($\mathcal{Q}_N$ and $\mathcal{Q}_D$ are made distinct)

Figure 3: LDBA for the formula $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,\mathsf{b}$.

(accepting component), i.e., $\mathcal{Q} = \mathcal{Q}_N \uplus \mathcal{Q}_D$, such that $q_0 \in \mathcal{Q}_N$ and $\mathcal{F} \subseteq \mathcal{Q}_D$. Subsets $\mathcal{Q}_N$ and $\mathcal{Q}_D$ are closed and deterministic under non-ε-transitions, i.e., $\delta(q, \sigma) \in \mathcal{Q}_D$ for all $q \in \mathcal{Q}_D$ and $\sigma \in \Sigma$, and $\delta(q, \sigma) \in \mathcal{Q}_N$ for all $q \in \mathcal{Q}_N$ and $\sigma \in \Sigma$. To transition from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ requires taking a non-deterministic ε-transition $\varepsilon \in \mathcal{E}$; taking an ε-transition does not consume an assignment, and thus ε-transitions can be thought of intuitively as *unobserved* jumps between two states. Note that there is at most one ε-transition for any path within the LDBA. Given an input ω-word w, a *run* $r = (q_0, q_1, q_2, \dots)$ of $\mathcal{B}$ is an infinite sequence of states $q \in \mathcal{Q}$ respecting the transition function δ. An ω-word w is *accepted* by $\mathcal{B}$ if there exists a run that infinitely often visits accepting states $q \in \mathcal{F}$.

Figure 3a presents an LDBA for the formula $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,\mathsf{b}$, which expresses a persistence requirement and a reachability specification. The automaton starts in state q_0 and transitions to the accepting state q_1 upon observing proposition b. Once it has reached q_1, it stays there indefinitely. This captures the $\mathsf{F}\,\mathsf{b}$ (eventually satisfy b) component of the specification. Alternatively, it can transition to the accepting state q_2 without consuming any input via the ε-transition ε_{q_2}. Once in q_2, the automaton accepts exactly the words where a is true at every step. This captures the $\mathsf{F}\,\mathsf{G}\,\mathsf{a}$ (eventually *always* satisfy a) component of the specification.

Note that Figure 3a presents a contracted (yet equivalent) version of the full LDBA that strictly follows the LDBA definition in Section 3. The full LDBA is illustrated in Figure 3b, with the initial component $\mathcal{Q}_N$ and accepting component $\mathcal{Q}_D$ made distinct. Note that this distinction is technically not possible in Figure 3a's contracted LDBA, yet the two automata are functionally equivalent since ε-transition ε_{q_4} is trivial; in other words, the contracted LDBA simply encodes that the automaton should *always* take ε-transition ε_{q_4} if possible.

This *LDBA contraction* is a practical optimization that removes all unnecessary ε-transitions, i.e., those corresponding to subformulae that do not require non-determinism (such as $\mathsf{F}\,\mathsf{b}$), and is done to reduce the number of nodes and edges while keeping LDBA behavior functionally equivalent. This helps to keep the reach-avoid sequences given to the agent as simple as possible, and also simplifies checking LTL satisfaction at run-time.

D Product MDPs

Product MDPs are defined as follows:

Definition D.1 (Product MDP; Jackermeier & Abate, 2025)**.** The *product MDP* $\mathcal{M}^\varphi$ of an MDP $\mathcal{M}$ and an LDBA $\mathcal{B}^\varphi$ synchronizes the execution of $\mathcal{M}$ and $\mathcal{B}^\varphi$. Formally, $\mathcal{M}^\varphi$ is an MDP with state space $\mathcal{S}^\varphi = \mathcal{S} \times \mathcal{Q}^\varphi$, action space $\mathcal{A}^\varphi = \mathcal{A} \cup \mathcal{E}^\varphi$, initial state distribution $\mu^\varphi(s, q^\varphi) = \mu(s) \cdot \mathbb{1}[q^\varphi = q_0^\varphi]$, and state-transition distribution

$$p^\varphi((s', q'^\varphi) \mid (s, q^\varphi), a) = \begin{cases} p(s' \mid s, a) & \text{if } a \in \mathcal{A} \wedge q'^\varphi = \delta^\varphi(q^\varphi, L^\varphi(s)), \\ 1 & \text{if } a = \varepsilon_{q'^\varphi} \wedge q'^\varphi = \delta^\varphi(q^\varphi, a) \wedge s' = s, \\ 0 & \text{else.} \end{cases}$$

The action space in $\mathcal{M}^\varphi$ is extended with $\mathcal{E}^\varphi$ to allow the policy to select ε-transitions in $\mathcal{B}^\varphi$ without acting in the MDP. Trajectories in $\mathcal{M}^\varphi$ are infinite sequences $\tau^\varphi = \big((s_0, q_0^\varphi), a_0, (s_1, q_1^\varphi), a_1, \dots\big)$, and the run $\tau_q^\varphi = (q_0^\varphi, q_1^\varphi, q_2^\varphi, \dots)$ is the projection of τ^φ onto $\mathcal{B}^\varphi$.

E ASIDE: CONNECTION TO PLATO'S THEORY OF FORMS

Anecdotally, it is often challenging to construct an LTL specification that accurately matches the desired task outcome, in particular those tasks originally defined in natural language. This is because natural language tasks tend to implicitly assume constraints, dependencies, and edge cases that are imperfectly translated (or forgotten entirely) during the formulation of the LTL specification.

We can draw a connection to Plato's Theory of Forms (Plato, 1992), wherein the ultimate "Form of the Good" (desired task outcome) is imperfectly represented as a composition of base concepts, or "Forms" (atomic predicates), which are instantiated as "particulars" (atomic propositions as predicate instances) to create an imperfect imitation of "Good" (the LTL specification).

While not scientifically relevant, we found this connection amusing and it inspired the name of our approach, and we believe it to be worthy of note.

F CONNECTION TO FIRST-ORDER LTL

First-order LTL (FOLTL; Gigante, 2025) extends LTL with atomic predicates over variables (defined over some domain D), as well as $\exists$ (exists) and $\forall$ (for-all) quantifiers that operate over these variables. Unlike the parameterized specifications considered in this paper, the variables in FOLTL are not necessarily instantiated with specific values within a given specification.

Example F.1. The FOLTL formula $\mathsf{G}(\exists\, x, y, z\, [\mathsf{pos}(x, y, z) \wedge \mathsf{gt}(z, 0.0)])$ represents that the agent position must remain above the $z = 0$ plane for all time. Literally, the formula can be interpreted as "always, there exists an assignment of position variable values such that 1) the values match the agent position, and 2) the z-value is greater than 0".

Example F.2. The FOLTL formula $\forall\, r, g, b\, [\mathsf{G}(\mathsf{at}(r, g, b) \Rightarrow F(\mathsf{at}(b, g, r)))]$ represents that whenever the agent enters a colored zone, it must eventually enter a colored zone with the R- and B-values swapped. Literally, the formula can be interpreted as "for all assignments of color variable values, whenever the agent enters a colored zone matching the color values, it must eventually enter a colored zone matching the same color values but with r and b swapped".

To ground FOLTL specifications in an MDP, each predicate f requires a labeling function $L_{\mathsf{f}}(s, x_{\mathsf{f}}) : \mathcal{S} \times \mathcal{X}_{\mathsf{f}} \to \{0, 1\}$, which returns its truth value for a given state $s \in \mathcal{S}$ and variable values $x_{\mathsf{f}} \in \mathcal{X}_{\mathsf{f}}$.

Unfortunately, FOLTL is undecidable in general (Gigante, 2025), and thus arbitrary FOLTL formulae cannot be translated into finite automata.[3] However, under the following restrictions, the logic reduces to LTL:

1. The $\exists$ and $\forall$ quantifiers are removed.
2. All variables are instantiated with fixed values within a given specification.

Under the first restriction, the syntax reduces to predicates and LTL's logical and temporal operators only. Under the second restriction, the predicates become *predicate instances*, which are atomic propositions; given fixed parameters x_{f}, the labeling function for each predicate instance becomes $L_{\mathsf{f}(x_{\mathsf{f}})}(s) : \mathcal{S} \to \{0, 1\}$, and the labeling function across all predicate instances in a specification φ becomes $L^{\varphi}(s) : \mathcal{S} \to \Sigma^{\varphi}$. Thus, our parameterized specifications belong to a fragment of FOLTL that behaves exactly like LTL from a logic perspective; the only difference from standard LTL is the replacement of propositions with predicate instances, which is purely notational.

G EXAMPLE BACKBONE: STRUCTLTL

This appendix outlines core details regarding the StructLTL backbone, introduced by Jackermeier et al. (2026), with the purpose of illustrating how a typical backbone interacts with our problem setting (Section 4), the Plato embedding module (Section 5), and P^{2}BRS (Section 6).

[3] Recent work by Olivieri et al. (2026) trains policies to satisfy specifications of LTLfMT, a finite-trace fragment of FOLTL modulo theories, but the method is single-task and restricted to finite-horizon specifications.

G.1 Reward Function

StructLTL trains a sequence-conditioned policy $\pi_\beta(a \mid s, \boldsymbol{\beta}_\pm)$. Given an n-step reach-avoid sequence $\boldsymbol{\beta}_\pm = \big((\beta_i^+, \beta_i^-)\big)_{i=0}^{n-1}$, StructLTL assigns a reward $r^\varphi = +1$ if the agent satisfies β_0^+ and $n = 1$; it assigns $r^\varphi = -1$ if the agent satisfies β_0^-; it assigns $r^\varphi = 0$ otherwise. Furthermore, the sequence is progressed to $\big((\beta_i^+, \beta_i^-)\big)_{i=1}^{n-1}$ if the agent satisfies β_0^+, keeping the task Markovian. By training over a distribution of such sequences, StructLTL learns a policy that generalizes across sequences, and therefore across LDBAs (via the sequences extracted from depth-first search).

This *sparse* reward function r^φ is used to approximately solve Equation 1 by training a policy that can execute paths to accepting states in an LDBA, thus implicitly maximizing LTL satisfaction probability by maximizing the number of visits to accepting states as a proxy. However, the trained policy is *biased* towards earlier satisfaction due to the discount factor γ.

Thus, when we claim that the augmented reward function $r^\varphi_{\text{aug}}(s, q^\varphi, a, s', q'^\varphi) = r^\varphi + \gamma\Phi(s', q'^\varphi) - \Phi(s, q^\varphi)$ under P^{2}BRS does not bias the optimal policy, this is with respect to the optimal policy under the *backbone's* reward function r^φ, not the hypothetical policy that maximizes Equation 1.

G.2 Model Architecture

To compute an embedding $\mathbf{z}_{\boldsymbol{\beta}_\pm}$ for $\boldsymbol{\beta}_\pm$, StructLTL first computes an embedding $\mathbf{z}_\beta$ for each constituent Boolean formula β. Note that each Boolean formula in $\boldsymbol{\beta}_\pm^\varphi = \text{DFS}_{\mathcal{B}^\varphi}(q^\varphi)$ is in disjunctive normal form (DNF), i.e., $\beta = \bigvee_{k=1}^{n_\beta} \bigwedge_{j=1}^{m_{\beta k}} l_{kj}$, where all $l_{kj} \in \{\mathsf{p}, \neg\mathsf{p}\}$ and $\mathsf{p} \in AP^\varphi$; see Appendix J for details. StructLTL exploits DNF structure by building clause embeddings from proposition embeddings, and then building disjunction embeddings from those clause embeddings:

$$\mathbf{z}_\beta = f_\vee\left(\sum_{k=1}^{n_\beta} f_\wedge\left(\sum_{j=1}^{m_{\beta k}} \Pi\big(\phi(\mathsf{p}_{kj})\big)\right)\right),$$

where ϕ is a learned encoder that maps proposition tokens to embeddings, Π denotes a learned linear projection if l_{kj} is negated and the identity function otherwise, and $f_\vee$ and $f_\wedge$ are learned networks applied after sum pooling, such that each level forms a permutation-invariant DeepSets encoder (Zaheer et al., 2017). An empty set (i.e., a clause with no literals, or a formula with no clauses; see Appendix J.2) is mapped to a learned embedding in place of the sum. We see that StructLTL satisfies Assumption 5.1, and the learned mapping ϕ can be replaced with the Plato embedding module.

The embeddings for each β_i^+ and β_i^- are concatenated to obtain the embedding $\mathbf{z}_i = \mathbf{z}_{\beta_i^+} || \mathbf{z}_{\beta_i^-}$ for each sequence step. The sequence embedding is then computed as that of the current subgoal contextualized by future subgoals; $\mathbf{z}_{\boldsymbol{\beta}_\pm} = \text{LayerNorm}(\mathbf{z}_0 + \mathbf{h})$ (Ba et al., 2016), where $\mathbf{h}$ is computed using single-head dot-product attention (Vaswani et al., 2017):

$$\mathbf{h} = \left[\text{softmax}\left(\frac{\mathbf{q}K^\top}{\sqrt{d_k}}\right)V\right]W^O,$$

where $\mathbf{q} = \mathbf{z}_0 W^Q$ is the query vector, W^O is the output projection matrix, and $K, V \in \mathbb{R}^{n \times d_k}$ are the key and value matrices obtained by projecting $\mathbf{z}_{0:n-1}$ using W^K and W^V, respectively. StructLTL employs ALiBi (Press et al., 2022) to bias the attention toward earlier subgoals, subtracting a linear penalty $m \cdot i$ from each unnormalized attention score before applying the softmax, with m being a fixed hyperparameter.

Finally, StructLTL concatenates $\mathbf{z}_{\boldsymbol{\beta}_\pm}$ with a learned encoding of the environment MDP state and passes this into an *actor-critic* MLP to obtain a distribution over actions and a predicted value. To handle ε-transitions, the policy's action space is augmented with a special ε-action which triggers an ε-transition in the LDBA state without performing an action in the environment MDP. For discrete action spaces, the policy network outputs a categorical distribution with an additional logit for the ε-action, while for continuous action spaces the policy network outputs a mixed discrete-continuous distribution. Thus, the policy must learn to predict *when* to trigger the ε-transition. The probability of selecting the ε-action is set to 0 when no ε-transitions are available for the current LDBA state.

H POLICY INVARIANCE UNDER P^{2}BRS

P^{2}BRS relies on the classic policy invariance of PBRS (Ng et al., 1999), which we restate over product states and prove for completeness:

Theorem H.1. *Given a bounded function $\Phi(s, q^\varphi)\colon \mathcal{S} \times \mathcal{Q}^\varphi \to \mathbb{R}$, the optimal policy under the augmented reward function $r^\varphi_{\text{aug}} = r^\varphi + \gamma\Phi(s', q'^\varphi) - \Phi(s, q^\varphi)$ is also optimal under r^φ.*

Proof. Consider the discounted return under $r^\varphi_{\text{aug}} = r^\varphi + \gamma\Phi(s', q'^\varphi) - \Phi(s, q^\varphi)$:

$$\begin{aligned} R^\varphi_{\text{aug}}(\tau^\varphi) &= \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma^t r^\varphi_t + \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma^{t+1}\Phi(s_{t+1}, q^\varphi_{t+1}) - \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma^t \Phi(s_t, q^\varphi_t) \\ &= R^\varphi(\tau^\varphi) + \lim_{T\to\infty} \sum_{t=1}^{T} \gamma^t \Phi(s_t, q^\varphi_t) - \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma^t \Phi(s_t, q^\varphi_t) \\ &= R^\varphi(\tau^\varphi) + \lim_{T\to\infty} \left(\gamma^T \Phi(s_T, q^\varphi_T)\right) - \Phi(s_0, q^\varphi_0) \\ &= R^\varphi(\tau^\varphi) - \Phi(s_0, q^\varphi_0), \end{aligned}$$

where the limit vanishes since Φ is bounded. The offset $\Phi(s_0, q^\varphi_0)$ does not depend on the actions taken, so it shifts the expected return of every policy equally (and, applied from any product state, shifts its value by $\Phi(s, q^\varphi)$), leaving the set of optimal policies unchanged. □

Finite Episodes. When an episode terminates at $T < \infty$, the same telescoping yields $R^\varphi_{\text{aug}}(\tau^\varphi) = R^\varphi(\tau^\varphi) + \gamma^T\Phi(s_T, q^\varphi_T) - \Phi(s_0, q^\varphi_0)$, where the residual term depends on the trajectory and would bias the optimal policy. The terminal override $\Phi(s_T, q^\varphi_T) = 0$ (Section 6.2) eliminates it, so the result of Theorem H.1 extends to finite episodes.

Interaction with Eventual Discounting. Some backbones may employ *eventual discounting* for their reward functions, only discounting time steps corresponding to visits to accepting LDBA states (Voloshin et al., 2023). In such cases, eventual discounting must *not* be applied to the potential difference component of the augmented reward, which requires standard discounting for the return to telescope correctly.

I PERSISTENCE GATING

This section gives the full definition of the Boolean formula robustness metric of Section 6.1.2. For a formula β satisfying Assumption 6.1, let $J^+_k = \{\, j : \mathsf{f}_{kj} \in \mathit{Pred}^+ \,\}$ index the persistent literals of clause c_k and $J^-_k = \{1{:}m_{\beta k}\} \setminus J^+_k$ the rest, where f_{kj} is the predicate of literal l_{kj}. With the conventions $\min_{j\in\emptyset}(\cdot) = 1$ and $\max_{k\in\emptyset}(\cdot) = -1$, we compute

$$\rho_{l_{kj}}(s) = \begin{cases} \rho_{\mathsf{f}}(s, x_{\mathsf{f}}) & \text{if } l_{kj} = \mathsf{f}(x_{\mathsf{f}}), \\ -\rho_{\mathsf{f}}(s, x_{\mathsf{f}}) & \text{if } l_{kj} = \neg\mathsf{f}(x_{\mathsf{f}}), \end{cases} \qquad h_k(s) = \min_{j\in J^+_k} \rho_{l_{kj}}(s), \quad t_k(s) = \min_{j\in J^-_k} \rho_{l_{kj}}(s),$$

$$\rho_{c_k}(s) = \begin{cases} h_k(s) \wedge_\sigma t_k(s) & \text{if } \emptyset \neq J^+_k \neq \{1{:}m_{\beta k}\}, \\ \min\big(h_k(s), t_k(s)\big) & \text{otherwise}, \end{cases} \qquad \rho_\beta(s) = \max_{k\in 1:n_\beta} \rho_{c_k}(s).$$

Negated Literals. Persistence is a property of the predicate rather than the literal: the truth value of a persistent predicate typically changes only through actions directed at it (e.g., grasping or releasing an object), so either value, once attained, tends to carry over while the agent works towards other literals. Gating therefore applies equally to negated literals; for example, for the clause $\neg\mathsf{has}(\mathit{obj}) \wedge \mathsf{pos}(x, y, z)$, the agent is guided to first release the object and then move to the target position, remaining empty-handed along the way. Other orderings may also satisfy the clause, such as carrying the object to the target and releasing it there, since releasing does not move the gripper. Gating need not identify the only valid ordering, but rather one that is simple and reliable to guide towards; as with any potential function, it shapes exploration without restricting the policy to which the agent ultimately converges.

Ungated Clauses. When $J_k^+ = \emptyset$ or $J_k^- = \emptyset$, the empty-set convention makes one argument of $\wedge_\sigma$ identically 1, so the staged conjunction would compress the whole unsatisfied range of the clause into a single stage's slice. We therefore fall back to the plain $\min$ over all literals of c_k.

Satisfaction. Since $|\rho_{\mathsf{f}}| > 0$, the staged conjunction is negative whenever either argument is negative, and positive otherwise, so $\rho_\beta(s) > 0$ if and only if β holds, mirroring $L_{\mathsf{f}} = \mathcal{H}(\rho_{\mathsf{f}})$ for individual predicates. Moreover, ρ_β remains in $[-1, 1]$, keeping Φ bounded.

J Reach-Avoid Sequence Extraction

This appendix details the Boolean reach-avoid sequence extraction procedure abstracted as $\boldsymbol{\beta}_\pm^\varphi = \mathrm{DFS}_{\mathcal{B}^\varphi}(q^\varphi)$ in the main text. The full procedure was first introduced by Jackermeier et al. (2026) but is outlined again here for clarity. The procedure comprises three stages: extraction of all accepting runs from q^φ (Appendix J.1), followed by construction of $\boldsymbol{\beta}_\pm^\varphi$ in minimal disjunctive normal form (Appendix J.2), and finally selection of the optimal sequence (Appendix J.3).

J.1 Computing Paths to Accepting Cycles

The DFS algorithm to compute all paths to accepting cycles in $\mathcal{B}^\varphi$ from q^φ, i.e., all accepting runs from q^φ, is presented in Algorithm 1. Note that the reach-avoid sequences are infinite, and so we approximate the looping part of the accepting cycle by truncation such that the truncated sequence visits an accepting state k times, where k is a hyperparameter (set to $k = 2$ for our experiments).

Algorithm 1 Computing paths to accepting cycles (Jackermeier & Abate, 2025)

Require: An LDBA $\mathcal{B} = \langle \mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E} \rangle$ and current state q

function DFS(q, p, i) ▷ i is the index of the last seen accepting state, or -1 otherwise
  $P \leftarrow \emptyset$
  if $q \in \mathcal{F}$ **then**
    $i \leftarrow |p|$
  end if
  for all $\sigma \in \Sigma \cup \mathcal{E}$ **do**
    $p' \leftarrow [p, q]$
    $q' \leftarrow \delta(q, \sigma)$
    if $q' \in p$ **then**
      if INDEX(q', p) $\leq i$ **then**
        $P \leftarrow P \cup \{p'\}$
      end if
    else
      $P \leftarrow P \cup$ DFS(q', p', i)
    end if
  end for
  return P
end function
$i \leftarrow \begin{cases} 0 & \text{if } q \in \mathcal{F} \\ -1 & \text{else} \end{cases}$
return DFS($q, [], i$)

J.2 Boolean Formula Construction

For each n-step prefix of an accepting run $\mathbf{q}^\varphi = (q_i^\varphi)_{i=0}^n$, where $q_0^\varphi = q^\varphi$, we construct a reach-avoid sequence of Boolean formulae $\boldsymbol{\beta}_\pm^\varphi = \big((\beta_i^{\varphi+}, \beta_i^{\varphi-})\big)_{i=0}^{n-1}$ such that

$$\forall \sigma^\varphi \in \Sigma^\varphi : \sigma^\varphi \models \beta_i^{\varphi+} \iff \delta^\varphi(q_i^\varphi, \sigma^\varphi) = q_{i+1}^\varphi,$$
$$\forall \sigma^\varphi \in \Sigma^\varphi : \sigma^\varphi \models \beta_i^{\varphi-} \iff \delta^\varphi(q_i^\varphi, \sigma^\varphi) \notin \{q_i^\varphi, q_{i+1}^\varphi\}.$$

In other words, $\beta_i^{\varphi+}$ represents the set of assignments that trigger the transition from q_i^φ to q_{i+1}^φ, while $\beta_i^{\varphi-}$ represents the set of assignments that trigger a transition to any other state (excluding self-loops). Sequence $\boldsymbol{\beta}_\pm^\varphi$ is thus a reach-avoid sequence to achieve accepting run $\mathbf{q}^\varphi$.

The construction is done using the Quine-McCluskey algorithm (Quine, 1952; McCluskey, 1956) to obtain each $\beta_i^{\varphi+}$ and $\beta_i^{\varphi-}$ in minimal disjunctive normal form (DNF). Given a set of satisfying assignments $S \subseteq \Sigma^\varphi$ and a set of non-satisfying assignments $N \subseteq \Sigma^\varphi$, the algorithm produces a Boolean formula that evaluates to true for assignments $\sigma^\varphi \in S$ and to false for assignments $\sigma^\varphi \in N$. The produced formula β is *DNF* because it is a disjunction of clauses of possibly negated propositions, i.e., $\beta = \bigvee_{k=1}^{n_\beta} \bigwedge_{j=1}^{m_{\beta k}} l_{kj}$, where all $l_{kj} \in AP^\varphi \cup \{\neg \mathsf{p} : \mathsf{p} \in AP^\varphi\}$. The formula is also *minimal* in the number of clauses and, among those, the number of literals. For $\beta_i^{\varphi+}$ we set $S = \{\sigma^\varphi \in \Sigma^\varphi : \delta^\varphi(q_i^\varphi, \sigma^\varphi) = q_{i+1}^\varphi\}$, for $\beta_i^{\varphi-}$ we set $S = \left\{\sigma^\varphi \in \Sigma^\varphi : \delta^\varphi(q_i^\varphi, \sigma^\varphi) \notin \{q_i^\varphi, q_{i+1}^\varphi\}\right\}$, and for both cases we use $N = \Sigma^\varphi \setminus S$. The degenerate cases $S = \emptyset$ and $S = \Sigma^\varphi$ yield the trivial formulae $\bot$ and $\top$, respectively; we treat $\bot$ as a degenerate DNF with no clauses ($n_\beta = 0$), and we treat $\top$ as a degenerate DNF with a single clause of no literals ($n_\beta = 1$, $m_{\beta 1} = 0$). In particular, $\beta_i^{\varphi-} = \bot$ whenever no transition from q_i^φ leads to a state other than q_i^φ or q_{i+1}^φ, i.e., there is nothing to avoid. Under the conventions of Appendix I, their bounded robustness metrics are $\rho_\top = 1$ and $\rho_\bot = -1$, corresponding to $g = \pm\infty$.

If a run $\mathbf{q}^\varphi$ requires an ε-transition from q_i^φ to q_{i+1}^φ, we set $\beta_i^{\varphi+} = \varepsilon$ in the corresponding reach-avoid sequence, where ε is a shared token used to represent any ε-transition. While there may be multiple ε-transitions in a given LDBA, there is at most one ε-transition for any path within the LDBA, and thus the rest of the "reach" component of $\boldsymbol{\beta}_\pm^\varphi$ gives sufficient context to uniquely identify its ε-transition. Meanwhile, we ignore any ε-transitions from q_i^φ to any other state, representing that the agent should never choose to take those ε-transitions.

In practice, both the DFS (run from every automaton state) and the Boolean formula construction can be performed alongside LDBA construction prior to training, yielding a cached set of reach-avoid sequences for every state in each formula structure's LDBA.

J.3 Optimal Sequence Selection

The DFS may yield a set of multiple reach-avoid sequences, $RA_{q^\varphi} = \{\boldsymbol{\beta}_\pm^\varphi\}_{q^\varphi}$, requiring us to choose among them upon changing LDBA state. We note that for backbones such as StructLTL, the policy's learned value function (or "critic") is an approximate lower bound on the (discounted) probability of LTL satisfaction, and thus choosing the reach-avoid sequence that maximizes critic value also maximizes (predicted) LTL satisfaction; see Jackermeier & Abate (2025) for details. Thus, our heuristic for sequence selection chooses the sequence that is predicted to maximize value.

To obtain a sequence for P^{2}BRS at training time, we select the sequence that maximizes the sequence-conditioned potential function Φ_β, i.e., $\boldsymbol{\beta}_\pm^{\varphi*} = \arg\max_{\boldsymbol{\beta}_\pm^\varphi \in RA_{q^\varphi}} \Phi_\beta(s, \boldsymbol{\beta}_\pm^\varphi)$; this is because the potential function acts as an a priori initialization of the value estimates (Wiewiora, 2003). At evaluation time, given the sequence-conditioned policy $\pi_\beta(a \mid s, \boldsymbol{\beta}_\pm)$, StructLTL selects the optimal sequence $\boldsymbol{\beta}_\pm^{\varphi*} = \arg\max_{\boldsymbol{\beta}_\pm^\varphi \in RA_{q^\varphi}} V^{\pi_\beta}(s, \boldsymbol{\beta}_\pm^\varphi)$, where V^{π_β} is the policy's value function. In both cases, a new sequence is only selected upon episode reset or changing LDBA state, i.e., $q'^\varphi \neq q^\varphi$, to ensure that the policy can commit to a sequence once selected.

K Experimental Details

K.1 Implementation

Our implementation is built in JAX (Bradbury et al., 2018). We use MuJoCo XLA (MJX; Todorov et al., 2012) for robotics simulation with models from the MuJoCo Menagerie (Zakka et al., 2022), and we use Rabinizer 4 (Křetínský et al., 2018) to convert LTL specifications into LDBAs. All experiments were run on an NVIDIA GeForce RTX 5090 GPU.

K.2 ENVIRONMENTS

K.2.1 GO1RGBZONEENV

Go1RGBZoneEnv expands upon *ZoneEnv* (Vaezipoor et al., 2021), requiring an MJX-simulated Unitree Go1 quadruped robot to navigate between colored zones. The arena is a square of side $6.6\,\mathrm{m}$ containing eight colored circular zones of radius $0.4\,\mathrm{m}$: four colors, with two zones per color. At the start of each episode, the zone centers are sampled uniformly over the central region of the arena, at least $1.1\,\mathrm{m}$ apart, and the robot is initialized in its standing pose at a uniformly sampled position at least $1.1\,\mathrm{m}$ from every zone center, with a uniformly sampled heading. The episode terminates if the robot leaves the arena or falls, i.e., its torso rolls past horizontal or its height drops below $0.10\,\mathrm{m}$; episodes are otherwise truncated after 1000 policy steps ($60\,\mathrm{s}$ of simulated time) during training. See Figure 5a for visualizations of example reset configurations.

Observations & Actions. The observation comprises the torso-frame planar velocity, the yaw rate, the last commanded forward speed, and a 32-bin RGB LiDAR. The LiDAR partitions bearings relative to the robot's heading into evenly spaced angular bins; each bin returns the color (r, g, b) of the nearest zone whose center lies within it, an intensity e^{-kd} (with $k = 0.5\,\mathrm{m}^{-1}$) that decays with the distance d to that zone's center, and a detection flag, with empty bins reading zero. All components are normalized to $[-1, 1]$. The action is a two-dimensional velocity command: forward speed $v_x \in [-1.5, 1.5]\,\mathrm{m\,s}^{-1}$ and yaw rate $\omega \in [-1.5, 1.5]\,\mathrm{rad\,s}^{-1}$.

Low-Level Control. The velocity command is tracked by a pre-trained locomotion controller, which is frozen during task training and executed deterministically. The controller is an MLP with hidden layers $(512, 256, 128)$, trained with PPO (Schulman et al., 2017) on a velocity-tracking joystick task whose hyperparameters and reward terms follow MuJoCo Playground's Go1 joystick task (Zakka et al., 2025), over a command envelope of $\pm 1.8\,\mathrm{m\,s}^{-1}$ and $\pm 2.0\,\mathrm{rad\,s}^{-1}$. It maps a 47-dimensional proprioceptive input — torso-frame linear and angular velocity, the projected gravity direction, joint angles relative to the standing pose, joint velocities, the previous controller action, and the current command — to position targets for the twelve joints, expressed as offsets from the standing pose. The physics timestep is $4\,\mathrm{ms}$ and the controller runs at $50\,\mathrm{Hz}$; each policy step holds the command for three controller steps, inducing a $0.06\,\mathrm{s}$ task-level control period. Although the task-level policy does not observe the Go1's joint-level proprioceptive state, the closed-loop behavior of the low-level controller forms part of the environment dynamics, which the policy must learn to account for through its partial observation of the torso state.

Predicates. The environment provides a single predicate `at` with color parameters $(r, g, b) \in [0, 1]^3$; specifications may name up to four instances (matching the zone colors). A predicate instance $\texttt{at}(r, g, b)$ holds if and only if the robot's planar torso position lies within the radius of a zone whose color is within Euclidean distance 0.25 of (r, g, b). Its robustness metric is the signed distance to the boundary of the nearest such color-matching zone, normalized by the zone radius (positive inside the zone); if no zone matches, the metric is the constant -1. The predicate is annotated as *non-persistent*. The world's four zone colors are drawn jointly and uniformly over the color cube, subject to a pairwise Euclidean separation of at least 0.7; the specification's `at` predicate instances take their parameters from this draw, so every named color is realized by two zones in the world. Since the separation exceeds twice the labeling tolerance, the tolerance balls around distinct instances are disjoint, and a zone matches at most one instance.

K.2.2 X2DRONESENV

X2DronesEnv loosely expands upon *FlatWorld* (Voloshin et al., 2023), requiring two MJX-simulated Skydio X2 quadcopters to navigate between spherical position targets in a shared airspace: a cube of side $20\,\mathrm{m}$ with no ground plane, in which the drones fly under gravity and aerodynamic drag. At the start of each episode, spherical position targets are placed throughout the airspace, and each drone is initialized at rest and level at a uniformly sampled position at least $5.5\,\mathrm{m}$ from every target center, with a uniformly sampled heading. The episode terminates if any drone leaves the airspace or if the two drones collide, i.e., their centers come within $0.72\,\mathrm{m}$ of one another (twice the radius of a sphere bounding each drone's propeller tips); episodes are otherwise truncated after 1000 policy steps ($50\,\mathrm{s}$ of simulated time) during training. See Figure 5b for visualizations of example reset configurations.

Observations & Actions. A single centralized policy controls both drones. The observation comprises, for each drone: its position, normalized to $[-1, 1]^3$; its attitude, as the first two columns of its rotation matrix (in the continuous 6D representation described by Zhou et al., 2019); its linear velocity, expressed in its heading frame (i.e., the world frame rotated by the drone's yaw) and normalized by the maximum commanded speed; and two proximity readings e^{-kc} on the clearance c to the nearest wall ($k = 1\,\mathrm{m}^{-1}$) and to the other drone ($k = 0.75\,\mathrm{m}^{-1}$). The action comprises, for each drone, a velocity command $(v_x, v_y, v_z) \in [-4, 4]^3\,\mathrm{m\,s}^{-1}$ and a yaw rate $\omega \in [-2, 2]\,\mathrm{rad\,s}^{-1}$, expressed in the same heading frame, so that a commanded v_x means the same horizontal direction whether or not the drone is tilting to accelerate.

The environment itself supports an arbitrary number of drones, but we deliberately study the two-drone configuration. In preliminary experiments with three drones, none of the methods we evaluated learned a strong policy. We attribute this to the joint observation and action spaces growing with the swarm while the policy remains a single centralized network with no weight sharing or permutation equivariance across drones, so the drone-to-target assignment implicit in wide conjunctions must be learned without any structural support — a setting more naturally addressed by multi-agent RL (MARL; Albrecht et al., 2024), which has been explored in combination with temporal-logic- and automaton-guided RL (Hammond et al., 2021; Neary et al., 2021). Such an extension is beyond the scope of this paper: two drones already suffice to exhibit the homogeneous conjunctions this environment is designed to study, and we leave swarm-scale variants to future work.

Low-Level Control. Each drone's velocity command is tracked by an identical model-based cascaded controller: a proportional velocity loop produces a desired thrust vector, whose tilt from vertical is capped at $30°$; a geometric attitude controller (Lee et al., 2010) computes the body torques that align the drone with this thrust vector; and the resulting thrust and torques are allocated to the four rotors within the actuator limits. The physics timestep is $10\,\mathrm{ms}$ and the controller runs at every physics step ($100\,\mathrm{Hz}$); each policy step holds the command for five physics steps, inducing a $0.05\,\mathrm{s}$ task-level control period. Although the policy does not observe the drones' angular velocities or rotor thrusts, the closed-loop behavior of the controller (i.e., its bounded tilt and its lag in tracking aggressive commands) forms part of the environment dynamics, which the policy must learn to account for through its observation of each drone's attitude and velocity.

Predicates. The environment provides a single predicate pos with position parameters $(x, y, z) \in [-10, 10]^3\,\mathrm{m}$. A predicate instance $\mathsf{pos}(x, y, z)$ holds if and only if some drone is within Euclidean distance $2\,\mathrm{m}$ of (x, y, z). Its robustness metric is the signed distance of the nearest drone to the boundary of the sphere, normalized by the sphere radius (positive inside the sphere). The predicate is annotated as *non-persistent*. The target positions are drawn jointly and uniformly over the central region of the airspace, subject to a pairwise separation of at least $5.5\,\mathrm{m}$; the specification's pos instances take their parameters directly from this draw. Since the separation exceeds twice the sphere radius, targets are disjoint: a single drone cannot satisfy two instances at once, and a conjunction $\mathsf{pos}_i \wedge \mathsf{pos}_j$ can only be satisfied by the two drones splitting across the corresponding spheres.

K.2.3 FRANKACUBESENV

FrankaCubesEnv loosely expands upon *WarehouseEnv* (Jackermeier et al., 2026), requiring an MJX-simulated Franka Panda manipulator arm with a parallel-jaw gripper to pick and place colored cubes of side length $5\,\mathrm{cm}$. The arm is commanded at its grasp center (the point between the fingertips), which is confined to a box-shaped workspace in front of the base. At the start of each episode, position and orientation targets for the grasp center are drawn, three cubes of distinct colors are placed on the floor at least $0.12\,\mathrm{m}$ apart with uniformly sampled yaws, and the arm is initialized at rest with the gripper open, at a uniformly sampled reachable pose at least $0.18\,\mathrm{m}$ from every position target, at least $0.8\,\mathrm{rad}$ from every orientation target, and clear of every cube. No condition terminates an episode early: the arm is confined to its workspace, and no collision is catastrophic. Episodes are truncated after 1000 policy steps ($50\,\mathrm{s}$ of simulated time) during training. See Figure 5c for visualizations of example reset configurations.

Observations & Actions. The observation comprises: the actual and commanded grasp-center poses, each as a normalized position and a 6D orientation (Zhou et al., 2019), such that the action displaces the *commanded* pose and the two differ by however far the joint servos lag; the arm's joint angles (as sines and cosines) and joint velocities; the finger opening, with flags reporting whether the gripper is driven open or closed, how long until it may be reversed, whether the last pose increment

was feasible, whether a cube is held, and whether a cube sits between the fingertip pads; a scene table listing each cube's position and color; and a pre-processed wrist camera looking along the gripper's approach axis (range $0.4\,\mathrm{m}$, half-angle $45°$) that reports the color of one cube (the held cube, or else the nearest cube in view), its offset from the grasp center, its relative orientation, and a detection flag. All components are normalized to $[-1, 1]$. The action comprises a Cartesian pose increment on the commanded grasp-center pose (effectively a twist command, given the fixed control period), of up to $2\,\mathrm{cm}$ of translation and $0.1\,\mathrm{rad}$ of rotation per step, together with a binary open/close gripper command; a change of gripper state is honored at most once every ten steps ($0.5\,\mathrm{s}$).

Low-Level Control. The commanded pose is resolved once per policy step into targets for the seven arm joints by closed-form inverse kinematics (He & Liu, 2021), which gravity-compensated joint position servos then track. The arm's redundancy is resolved by sweeping the final wrist joint about its current angle and taking the solution requiring the smallest joint motion; solutions moving any joint by more than $0.5\,\mathrm{rad}$ (e.g., across a singularity) are rejected, and if no admissible solution exists the previous command is held and the observation flags the increment as infeasible. The physics timestep is $5\,\mathrm{ms}$; each policy step holds the joint targets for ten physics steps, inducing a $0.05\,\mathrm{s}$ task-level control period. The fingers are driven by a position servo whose closing force is capped at $20\,\mathrm{N}$, so a closed gripper presses on whatever it meets with a fixed force. A grasp is registered when the closed fingers stall against a cube sitting squarely between the fingertip pads; the grasped cube is then held rigidly to the hand until the gripper opens, standing in for the stiction that MJX's velocity-level friction model lacks.

Predicates. The environment provides three predicates. A predicate instance $\mathsf{pos}(x, y, z)$, with position parameters (x, y, z) ranging over the workspace box $[0.25, 0.77] \times [-0.32, 0.32] \times [0.02, 0.50]\,\mathrm{m}$, holds if and only if the grasp center is within Euclidean distance $0.07\,\mathrm{m}$ of (x, y, z); its robustness metric is the signed distance to the boundary of that sphere, normalized by the sphere radius (positive inside the sphere). A predicate instance $\mathsf{rot}(\phi, \theta, \zeta)$, with roll–pitch–yaw parameters $(\phi, \theta, \zeta) \in [-\pi, \pi]^3$, holds if and only if the geodesic angle on $\mathrm{SO}(3)$ between the grasp-center orientation and the target orientation is within a tolerance of $0.3\,\mathrm{rad}$; its robustness metric is the tolerance minus this angle, normalized by the tolerance. A predicate instance $\mathsf{has}(r, g, b)$, with color parameters $(r, g, b) \in [0, 1]^3$, holds if and only if the gripper is holding a cube whose color is within Euclidean distance 0.25 of (r, g, b); its robustness metric is $+1$ when the instance holds, and otherwise the negative distance from the grasp center to the nearest cube of matching color; if no cube matches, the metric is the negative of the workspace diagonal. pos and rot are annotated as *non-persistent*; has is *persistent*, since a grasped cube travels with the gripper and so remains held through the motion that goes on to satisfy a pos or rot conjunct.

Position and orientation targets are drawn uniformly, subject to pairwise separations of $0.18\,\mathrm{m}$ and $0.8\,\mathrm{rad}$ respectively and to a joint reachability filter guaranteeing that every drawn position is reachable at every drawn orientation, since a specification may conjoin any pair; positions are drawn over the central region of the workspace, whose bottom face sits at a resting cube's center height. The world's three cube colors are drawn jointly and uniformly over the color cube, subject to a pairwise separation of at least 0.7; the specification's has instances take their parameters from this draw, so every named color is stocked by a cube. Since each separation exceeds twice the corresponding tolerance, distinct instances of the same predicate cannot hold simultaneously for the single gripper; satisfiable conjunctions instead mix predicates, e.g., $\mathsf{has}_i \wedge \mathsf{pos}_j \wedge \mathsf{rot}_k$, which requires holding a cube of a named color at a named position and orientation.

K.2.4 FALLOUTWORLD

FalloutWorld expands upon *LetterWorld* (Vaezipoor et al., 2021), requiring an agent to navigate a 21×21 grid overlaid with a radiation field that assigns each cell a level in $[0, 1]$. The agent occupies a single cell and steps between adjacent cells; specifications direct it to visit target cells in particular orders while avoiding, or seeking, radiation above a named threshold. At the start of each episode, the target cells and the radiation threshold are drawn, a fresh radiation field is generated around them, and the agent spawns at a below-threshold cell. No condition terminates an episode early: the grid is bounded, and radiation affects the labeling function rather than the dynamics. Episodes are truncated after 500 steps during training. See Figure 5d for visualizations of example reset configurations.

Observations & Actions. The observation comprises two top-down images: the full grid, with one channel marking the agent's cell and one carrying the radiation field; and a 9×9 crop of the radiation

field centered on the agent, paired with a mask marking which of its cells lie on the grid (off-grid cells read zero), which presents nearby radiation in coordinates relative to the agent. All channels take values in $[0, 1]$. The action selects one of five discrete moves: a step to the adjacent cell in one of the four cardinal directions, or remaining in place; a step off the grid leaves the agent in place.

Predicates. The environment provides two predicates. A predicate instance $\mathsf{loc}(x, y)$, with cell parameters (x, y) ranging over the grid, holds if and only if the agent occupies the cell (x, y); its robustness metric is the negated Manhattan distance from the agent's cell to (x, y). A predicate instance $\mathsf{rad}(\eta)$, with threshold parameter $\eta \in [0.2, 0.8]$, holds if and only if the radiation level at the agent's cell is at least η; its robustness metric is the negated Manhattan distance to the nearest cell whose radiation level is at least η. Both predicates are annotated as *non-persistent*.

The target cells are drawn jointly and uniformly over the grid, subject to a pairwise Manhattan separation of at least 7 cells, and the specification's loc instances take their parameters directly from this draw; the agent's spawn cell is drawn with the same separation from every target, and the threshold uniformly over its range. Each target cell is then independently assigned, with equal probability, to lie above or below the threshold, and the radiation field is generated to realize this assignment. The field is a composition of smooth elliptical plateaus: ambient plateaus exceeding the threshold populate the grid, corrective plateaus force each target cell to its assigned side of the threshold, and decoy plateaus peaking below the threshold ensure that the presence of a plateau does not by itself reveal whether it exceeds the threshold. The generator guarantees that the spawn cell lies below the threshold, that every below-threshold target is reachable from the spawn without crossing the threshold (a below-threshold corridor is carved through the field where necessary), and that some cell exceeds the threshold, so specifications that avoid radiation and specifications that seek it remain feasible in every world. Since the agent occupies a single cell, distinct loc instances cannot hold simultaneously; a conjunction $\mathsf{loc}_i \wedge \mathsf{rad}$ is satisfiable in a given world if and only if target i is assigned above the threshold, and holds upon arrival there.

K.3 Evaluation LTL Specifications

Tables 4a and 4b list the template *finite*-horizon and *infinite*-horizon specifications, respectively. The evaluation specifications for *Go1RGBZoneEnv* are adapted from those used for evaluation in *ZoneEnv* by Jackermeier & Abate (2025). For the other environments, the specifications are new and designed to capture the complexity of the new environments, including conjunctions with the same predicates for *X2DronesEnv*, and conjunctions with different predicates for *FrankaCubesEnv*.

For *FrankaCubesEnv*, tasks φ_{22}–φ_{24} require placing cubes at target positions on the floor (pos_0 and pos_1). Accordingly, for those tasks we fix the z-parameter for pos_0 and pos_1 to 0.025, corresponding to the center offset height of a $5\,\mathrm{cm}$ cube when resting on the floor.

K.4 Training Details

K.4.1 Neural Networks

Proposition Embeddings. Plato's MLPs ϕ_{f} have a single hidden layer of size 32 with ReLU activation and a linear output layer of size n_ϕ. Discretize replaces them with one embedding table per predicate, holding a row of size n_ϕ for every cell of the discretized parameter space; parameters map to the nearest bin center under the corresponding domain's metric. The position and color parameter spaces of pos, at, and has are discretized to 11 bins per dimension ($11^3 = 1331$ rows each); the orientation space of rot in *FrankaCubesEnv* to 2048 orientations selected by farthest-point sampling under the geodesic distance; and in *FalloutWorld*, loc ranges over all 441 grid cells and rad over 13 threshold bins. Concatenate and SIAMS instead retain learned proposition token embeddings of size n_ϕ for the backbone and route the parameters to the observation encoder (described below), gathered into a fixed-width block with one row per predicate instance in the alphabet, zero-padded to a common width, and with the rows of instances absent from the current sequence zeroed. Concatenate appends this block to the observation encoder's input (to the proprioceptive input for *Go1RGBZoneEnv*, and to the flattened convolutional outputs for *FalloutWorld*), while SIAMS passes it through a FiLM MLP with one hidden layer (of size 512 for *Go1RGBZoneEnv* and *FalloutWorld*, and 256 for *X2DronesEnv* and *FrankaCubesEnv*) that outputs an element-wise scale and offset applied to the observation encoder's output.

StructLTL. The sequence encoder for the StructLTL backbone (Appendix G) uses proposition embeddings of size $n_\phi = 32$, a learned vector of the same size embeds the ε-transition, and the negation projection Π is a linear layer of size 32. The post-pooling networks $f_\wedge$ and $f_\vee$ are each a single linear layer of size 32 with ReLU activation, and each has a separate learned embedding for the empty set. The reach and avoid embeddings of each sequence step are concatenated into a step embedding of size 64, and the attention mechanism projects queries, keys, and values to $d_k = 64$, with an ALiBi slope of $m = 5$.

Observation Encoders. For *Go1RGBZoneEnv*, the observation encoder uses a 1D CNN for the 32-bin RGB LiDAR observations, with $[32, 64]$ channels, a kernel size of 5, a stride of 1, circular padding, and an average-pooling layer with kernel size and stride 2 after each convolution; a linear layer of size 64 embeds the proprioceptive observations, and the concatenated outputs pass through a linear layer of size 512. For *X2DronesEnv* and *FrankaCubesEnv*, the observation encoder is an MLP with hidden layer sizes $[256, 256]$ and an output size of 256. For *FalloutWorld*, the observation encoder applies two independent 2D CNNs, one to the full-grid image, with $[8, 8, 2]$ channels, and one to the egocentric crop, with $[16, 16, 4]$ channels; both use 3×3 kernels with stride 1 and unit zero-padding except for a final 1×1 convolution, and their flattened outputs are concatenated. All observation encoders use ReLU internal activations.

Actor-Critic. For all environments, the actor and critic are MLPs with hidden layer sizes $[512, 512]$ and ReLU internal activations, operating on the concatenation of the observation embedding and the goal embedding from the backbone. In the continuous environments, the actor has a $\tanh$ output layer for the mean and standard deviation of a diagonal Gaussian distribution over the action space (the standard deviation then passes through a softplus activation to remain strictly positive), alongside a logit for the ε-action. For *FrankaCubesEnv*, whose action space pairs a continuous grasp-center pose command with a discrete gripper command, the actor adds a categorical output layer for the gripper, and the policy factorizes over the two distributions. In the discrete *FalloutWorld*, the actor has a softmax output layer for a categorical distribution of size $|\mathcal{A}| + 1$, where the additional logit represents the ε-action. The critic outputs the predicted value of the value function.

LTL2Action. The formula encoder for the LTL2Action backbone (Appendix N.2) uses proposition and operator token embeddings of size $n_\phi = 32$, and the syntax-tree encoder is an RGCN with 8 message-passing rounds, weights shared across rounds, three edge relation types, and $\tanh$ activations; the root node's embedding after the final round serves as the specification embedding. The observation encoders, actor, and critic match those above, except that the actor omits the ε-logit, since LTL progression admits no ε-transitions. The proposition embeddings under Plato and the baselines are produced as for StructLTL.

Model Size. For our environments, each MLP ϕ_{f} has 1120–1184 parameters, and the complete Plato embedding module between 1184 (*Go1RGBZoneEnv* and *X2DronesEnv*, one predicate each) and 3552 (*FrankaCubesEnv*, three predicates). The overall policy networks comprise between 1.0 M and 1.9 M parameters, dominated by the actor, critic, and observation encoders; the StructLTL sequence encoder accounts for roughly 20 000. The Plato embedding module thus makes up less than 0.4% of the overall policy network in every environment, in line with the claim of Section 5.

K.4.2 Hyperparameters

We use PPO (Schulman et al., 2017) for training. The hyperparameters for PPO and P^{2}BRS are listed in Table 2; all methods use the same hyperparameters. We use Adam (Kingma & Ba, 2015) for all experiments. The choice of RL algorithm is not critical to PlatoLTL; while other algorithms could be used, we opted for PPO as it is the algorithm originally used for the StructLTL and LTL2Action backbones, and it generally achieves fast wall-clock training time.

K.4.3 Curriculum

For the StructLTL backbone, we use *curriculum learning* to gradually introduce more challenging sequences during training, starting with one-step reach tasks and increasing to multi-step reach-avoid and reach-stay tasks by the final curriculum stage. Table 5 presents our generic 7-stage training curriculum shared by all environments. Reach-avoid sequences take the form $\big((\beta_i^+, \beta_i^-)\big)_{i=0}^{n-1}$, while reach-stay sequences take the form $\big((\varepsilon, \beta^-), (\beta^+, \neg\beta^+), (\beta^+, \neg\beta^+), \dots\big)$. The curriculum struc-

Table 2: Hyperparameters for PPO and P^{2}BRS. Values spanning multiple columns indicate identical values across those environments.

		Continuous			Discrete
	Parameter	Go1RGBZoneEnv	X2DronesEnv	FrankaCubesEnv	FalloutWorld
PPO	Number of parallel envs	2048			
	Steps per env per update	64			
	Epochs	5			4
	Batch size	32768			8192
	Discount factor	0.995			0.99
	GAE λ	0.95			
	Entropy coefficient	0.003			0.03
	Value loss coefficient	0.5			
	Max gradient norm	1.0			
	Clip ϵ	0.2			
	Adam learning rate	5×10^{-4}			
	Adam ϵ	1×10^{-8}			
	Total env interactions	1×10^8		5×10^8	1×10^8
	Maximum episode length	1000			500
P^{2}BRS	r_{rch}	0.4			
	r_{avd}	0.1			
	σ	0.5			
	q_g / q_κ	0.1			
	n_g / n_κ	131072			
	λ_κ	0.03			
	$\kappa_{\min}$	0.1			
	$\kappa_{\max}$	10.0			

ture is deliberate: stage 1 introduces reaching propositions, stage 2 extends this to clauses, stage 3 introduces a "narrow" set of clauses to avoid, stage 4 extends this to a "wide" set of disjunctions, stage 5 introduces multi-step sequences, stage 6 introduces reach-stay tasks, and stage 7 serves as the final training distribution. For environments without persistence ($\mathsf{F}\,\mathsf{G}$) tasks in the evaluation set, stages 6 and 7 are excluded from training. To progress to the next stage, success rate must be greater than or equal to 0.95, measured over a sliding window of the 512 most recent completed episodes.

Because LTL2Action (used for Appendix N.2) conditions the policy on the task formula directly, progressing it after each environment step, it cannot consume the reach-avoid sequences our standard curriculum produces; for *Go1RGBZoneEnv* and *X2DronesEnv* we use small bespoke curricula instead. These roughly mirror the standard ladder's structure. Tasks are sequential reach-avoid formulae of the form $\mathsf{F}\big(r_1 \wedge \mathsf{F}(r_2 \wedge \cdots)\big)$, where any $\mathsf{F}(r_i \wedge \cdot)$ may be replaced by $\neg a_i \mathsf{U}(r_i \wedge \cdot)$ to place a guard a_i on the i-th step. The subgoals r_i and guards a_i are sampled from propositions and disjunctions of propositions, plus conjunctions of propositions for *X2DronesEnv*, with increasing depth and number of disjunctions (and conjunctions) in later stages.

L Evaluation Results for Infinite-Horizon LTL Specifications

Table 3 presents the average number of visits for *infinite*-horizon LTL specifications. Note that specifications ψ_1–ψ_4 yield hundreds of visits on average; this is because these involve *persistence* ($\mathsf{F}\,\mathsf{G}$) tasks, and once the policy makes the ε-transition, it completes a visit at every timestep so long as the specification is not violated. The remaining infinite-horizon LTL specifications involve *recurrence* ($\mathsf{G}\,\mathsf{F}$) tasks, for which visits occur much more sparsely. Note also that ψ_{11}, ψ_{18}, ψ_{19}, and ψ_{24} have a constant offset of $+1$ to average visits; this is a quirk of the LDBAs constructed for these tasks by Rabinizer, which yield a visit upon the first automaton transition regardless of MDP state.

As with the *finite* tasks, we see strong performance on *infinite* tasks from Plato in *Go1RGBZoneEnv* and *FalloutWorld* with considerably poorer performance from the baselines, as well as non-zero visits by Plato in *X2DronesEnv* while the baselines make no progress. Applying P^{2}BRS to Plato significantly increases average visits in *X2DronesEnv*, and yields even more dramatic improvement in *FrankaCubesEnv*, achieving high numbers of average visits for tasks with otherwise near zero visits under unguided Plato. Again, we see little to no gains from P^{2}BRS in *Go1RGBZoneEnv* and *FalloutWorld*, with the exception of persistence tasks (ψ_1–ψ_4), which see a mild boost on average, likely because the dense signal guides the agent to stay within the colored zones.

Table 3: Evaluation results for *infinite*-horizon LTL specifications.

		Average Visits (↑)				
	ψ	Discretize	Concatenate	SIAMS	Plato	Plato + P²BRS
GoToRGBZoneEnv	ψ_1	$23.1^{+3.5}_{-2.2}$	$303.0^{+206.5}_{-148.6}$	$32.1^{+29.6}_{-9.1}$	$837.9^{+59.3}_{-57.3}$	$\mathbf{871.8}^{+76.6}_{-78.4}$
	ψ_2	$22.3^{+3.3}_{-1.9}$	$293.4^{+205.6}_{-145.7}$	$30.2^{+27.0}_{-8.1}$	$808.8^{+56.1}_{-59.6}$	$\mathbf{875.6}^{+81.9}_{-73.9}$
	ψ_3	$11.7^{+1.7}_{-0.9}$	$253.2^{+185.8}_{-127.6}$	$26.1^{+27.3}_{-8.2}$	$816.3^{+71.6}_{-53.8}$	$\mathbf{848.1}^{+82.1}_{-81.7}$
	ψ_4	$18.5^{+2.0}_{-1.4}$	$213.6^{+155.2}_{-109.5}$	$34.8^{+25.3}_{-7.8}$	$787.1^{+57.3}_{-65.9}$	$\mathbf{836.4}^{+73.4}_{-89.5}$
	ψ_5	$10.2^{+0.6}_{-0.7}$	$16.2^{+1.7}_{-1.0}$	$12.8^{+1.2}_{-1.7}$	$29.2^{+1.5}_{-2.1}$	$\mathbf{29.9}^{+0.8}_{-0.8}$
	ψ_6	$0.3^{+0.1}_{-0.1}$	$5.5^{+0.5}_{-0.5}$	$3.1^{+0.6}_{-0.6}$	$13.6^{+0.7}_{-0.8}$	$\mathbf{14.2}^{+0.3}_{-0.7}$
X2DronesEnv	ψ_7	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$4.0^{+0.2}_{-0.2}$	$\mathbf{4.5}^{+0.2}_{-0.2}$
	ψ_8	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$3.3^{+0.7}_{-1.0}$	$\mathbf{6.3}^{+0.3}_{-0.4}$
	ψ_9	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$5.6^{+0.8}_{-1.4}$	$\mathbf{8.0}^{+0.2}_{-0.4}$
	ψ_{10}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$1.5^{+0.5}_{-0.5}$	$\mathbf{5.0}^{+0.2}_{-0.3}$
	ψ_{11}	$1.0^{+0.0}_{-0.0}$	$1.0^{+0.0}_{-0.0}$	$1.0^{+0.0}_{-0.0}$	$3.8^{+0.5}_{-0.5}$	$\mathbf{4.7}^{+0.2}_{-0.2}$
	ψ_{12}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$1.6^{+0.5}_{-0.6}$	$\mathbf{5.1}^{+0.2}_{-0.2}$
FrankaCubesEnv	ψ_{13}	$0.1^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$19.6^{+1.9}_{-2.0}$	$\mathbf{29.7}^{+1.2}_{-1.7}$
	ψ_{14}	$0.1^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$14.6^{+4.5}_{-6.1}$	$\mathbf{33.9}^{+2.2}_{-1.6}$
	ψ_{15}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.7^{+0.3}_{-0.2}$	$\mathbf{12.9}^{+1.2}_{-0.7}$
	ψ_{16}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.2}_{-0.0}$	$\mathbf{39.4}^{+3.8}_{-5.8}$
	ψ_{17}	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$\mathbf{4.9}^{+1.0}_{-0.7}$
	ψ_{18}	$1.0^{+0.0}_{-0.0}$	$1.0^{+0.0}_{-0.0}$	$1.0^{+0.0}_{-0.0}$	$1.2^{+0.6}_{-0.2}$	$\mathbf{7.4}^{+0.4}_{-0.5}$
FalloutWorld	ψ_{19}	$27.7^{+0.4}_{-0.4}$	$6.2^{+1.0}_{-1.1}$	$1.0^{+0.0}_{-0.0}$	$\mathbf{30.8}^{+0.2}_{-0.2}$	$29.9^{+0.3}_{-0.2}$
	ψ_{20}	$7.0^{+0.4}_{-0.3}$	$0.4^{+0.1}_{-0.1}$	$0.0^{+0.0}_{-0.0}$	$\mathbf{15.4}^{+0.5}_{-0.5}$	$\mathbf{15.4}^{+0.4}_{-0.4}$
	ψ_{21}	$3.8^{+0.2}_{-0.3}$	$0.1^{+0.0}_{-0.0}$	$0.0^{+0.0}_{-0.0}$	$10.0^{+0.4}_{-0.4}$	$\mathbf{10.2}^{+0.3}_{-0.3}$
	ψ_{22}	$18.1^{+0.3}_{-0.3}$	$0.8^{+0.3}_{-0.2}$	$0.0^{+0.0}_{-0.0}$	$\mathbf{19.6}^{+0.1}_{-0.1}$	$19.2^{+0.1}_{-0.1}$
	ψ_{23}	$52.1^{+1.0}_{-0.8}$	$18.0^{+2.5}_{-2.0}$	$1.3^{+0.6}_{-0.4}$	$\mathbf{58.9}^{+0.5}_{-0.4}$	$57.4^{+0.7}_{-0.7}$
	ψ_{24}	$4.0^{+0.4}_{-0.4}$	$1.0^{+0.0}_{-0.0}$	$1.0^{+0.0}_{-0.0}$	$4.9^{+0.7}_{-0.5}$	$\mathbf{5.3}^{+0.5}_{-0.4}$

M Trajectory Visualizations

Figure 6 presents example trajectories for Plato with P²BRS evaluated on *finite* and *infinite* tasks, truncated to the first 4 subgoals.

N Ablation Studies

N.1 Principal Component Analysis of Proposition Embeddings

This ablation study tests our claim that the Plato embedding module learns low-dimensional per-predicate geometries in embedding space. We consider the two environments with multiple predicates: *FrankaCubesEnv* (with $\mathsf{pos}(x, y, z)$, $\mathsf{rot}(\phi, \theta, \zeta)$, and $\mathsf{has}(r, g, b)$) and *FalloutWorld* (with $\mathsf{rad}(\eta)$ and $\mathsf{loc}(x, y)$). We compare policies learned using Plato and Discretize (the other baselines do not learn unique embeddings for each predicate instance); we also include Plato with P²BRS in *FrankaCubesEnv*. For each environment, we sample 1024 predicate instances for each predicate and perform a series of principal component analyses (PCAs) on the corresponding learned proposition embeddings under each method. We perform a combined PCA over all predicates as well as per-predicate PCAs, measuring the explained variance ratios $\lambda_i \in [0, 1]$ for each component i ($\Sigma_i \lambda_i = 1$). For each predicate we also compute the *participation ratio* $\mathrm{PR} = {(\Sigma_i \lambda_i)^2}/{\Sigma_i \lambda_i^2}$, which measures the effective dimensionality of its embedding distribution.

Figure 7 presents the results for *FrankaCubesEnv*. We see that Plato (both with and without P²BRS) learns smooth, low-dimensional geometries for each predicate, cleanly separated in embedding space. For Plato with P²BRS, the distributions for pos and has have a PR of 2.6 and 2.5 respectively with a steep drop in explained variance ratio after the third component, corresponding to roughly 3D shapes (as we would expect given the three roughly independent input parameters for each). The distribution for rot yields a moderately higher PR of 4.8, with its PCA spectrum suggesting a flatter attenuation in explained variance ratio per component. This may be due to a more complex relationship between angles for reachable orientations (since we only sample reachable orientations). Plato without P²BRS shows very similar results except that the distribution for has is compressed to a PR of 1.1, with the dominant component aligned with luminance. This is possibly an artifact of unguided Plato failing to learn to satisfy tasks involving has predicate instances, which require the manipulator arm to pick up a correctly colored cube (the most difficult aspect of the environment). Since unguided Plato still yields low-dimensional per-predicate geometries despite failing to converge to a strong policy, we note that Plato's inductive bias is not wholly sufficient to solve complex robotics environments such as *FrankaCubesEnv*, with P²BRS needed to close the gap.

Meanwhile, we see significant overlap between the distributions for each predicate under Discretize. Furthermore, the per-predicate PRs are all in the range 29.1–30.1; given the embedding dimension $n_\phi = 32$, these distributions effectively span the entire high-dimensional embedding space. They are also highly non-smooth, with no discernible spatial correlations with respect to parameter values.

Figure 8 presents the results for *FalloutWorld*, for which we see the same trend, with Plato learning cleanly separated low-dimensional shapes for each predicate while Discretize learns non-smooth distributions spanning the embedding space. Plato yields a PR of 1.0 and 2.2 for rad and loc respectively, as is to be expected given their parameter input dimensions. In contrast, Discretize yields a PR of 28.9 for loc; the relatively low PR of 7.9 for rad is deceptive and is simply due to there only being 13 discrete bins for this predicate. Note that Discretize performs relatively well in *FalloutWorld* (at least compared to Concatenate and SIAMS), since the loc bins correspond exactly to the possible grid locations. Thus, even a "successful" policy under Discretize essentially memorizes a dictionary of independent embeddings rather than organizing them into coherent geometries for efficient learning. However, we do at least see some separation between predicates under Discretize for this environment.

N.2 PLATOLTL WITH LTL2ACTION

This study demonstrates that both the Plato embedding module and P^{2}BRS can be applied to backbones beyond StructLTL. We use LTL2Action (Vaezipoor et al., 2021), which encodes the syntax tree of the LTL specification using a GNN and updates the specification via LTL progression as the agent progresses through the task. We chose this backbone since it is widely influential within the field and considerably different from sequence-conditioned policies such as DeepLTL (Jackermeier & Abate, 2025), GenZ-LTL (Guo et al., 2025), and StructLTL (Jackermeier et al., 2026).

LTL2Action uses a learned encoder ϕ to map proposition tokens into embeddings, thus satisfying Assumption 5.1 cleanly and enabling the use of the Plato embedding module. However, to apply P^{2}BRS we require that the LTL2Action policy operates over the product MDP space $\mathcal{S} \times \mathcal{Q}^\varphi$ such that we can obtain $\Phi(s, q^\varphi) = \Phi_\beta\big(s, \mathrm{DFS}_{\mathcal{B}^\varphi}(q^\varphi)\big)$; indeed, we define a policy over product MDP states, $\pi(a \mid s, q^\varphi)$, as part of our problem setting (Section 4). While LTL2Action uses LTL progression rather than constructing an explicit LDBA, the two representations track the same underlying object: each progressed formula corresponds, up to propositional equivalence, to a state of the automaton obtained by the "after function" construction of Sickert et al. (2016), from which the LDBA is derived. Conversely, for the co-safe fragment of LTL admitted by LTL2Action, every reachable automaton state is represented by a progressed formula: co-safe automata require no accepting-component bookkeeping, as their accepting states are absorbing and represented by the formula $\top$. We therefore run an LDBA monitor alongside the executing policy and define $form(q^\varphi)$ *operationally* as the formula obtained by progressing φ along the trace that reached q^φ; by the correspondence above, this formula and the monitored LDBA state evolve in lockstep, so $form$ is well-defined along every execution without requiring a canonical representative per abstract state. This allows us to write the policy as $\pi(a \mid s, q^\varphi) = \pi_{form}\big(a \mid s, form(q^\varphi)\big)$, where π_{form} is LTL2Action's formula-conditioned policy, enabling us to compute Φ using the monitored automaton state q^φ.

Figure 9a presents evaluation curves during training for *finite* tasks in *Go1RGBZoneEnv* using LTL2Action. Much like we saw for StructLTL, Plato demonstrates strong performance, converging to a high success rate and low episode length while the baselines remain far from convergence. We see that converged success rate and time to converge are both notably worse than those when using StructLTL; we attribute this to the backbone itself, since StructLTL has been demonstrated to significantly outperform LTL2Action in modern benchmarks (Jackermeier et al., 2026). Interestingly, applying P^{2}BRS yields markedly faster convergence compared to unguided Plato when using LTL2Action, which was not the case for StructLTL in *Go1RGBZoneEnv*. This suggests that LTL2Action suffers an even greater exploration bottleneck than StructLTL, which P^{2}BRS alleviates.

Figure 9b presents evaluation curves for *finite* tasks in *X2DronesEnv*, over an extended training budget of 2.5×10^8 environment interactions. Unguided Plato's success rate flatlines alongside the baselines, a divergence in behavior from that observed with StructLTL, where Plato demonstrated steady learning progress. Again, we attribute this to the capability of the LTL2Action backbone. However, we do see clear training progress upon application of P^{2}BRS, with success rate rising slowly but steadily.

We omit results in *FrankaCubesEnv* since LTL2Action did not yield signs of learning progress even upon application of P^{2}BRS. Thus, it is essential to use a strong backbone in combination with PlatoLTL (e.g., StructLTL) to solve the most challenging environments and tasks; neither is independently sufficient.

N.3 P^{2}BRS Ablations

This study ablates key design choices in P^{2}BRS, justifying the need for persistence gating and adaptive robustness scale calibration, as well as investigating the effect of replacing P^{2}BRS's hard $\min/\max$ aggregators with smooth combinators. We also consider automaton-based PBRS as a baseline, demonstrating its insufficiency for multi-task RL in high-dimensional environments with complex subgoals. Figure 10 presents evaluation curves during training for *finite* tasks in *X2DronesEnv* and *FrankaCubesEnv* under each ablation, as well as Plato with and without P^{2}BRS for reference.

We do not include Ground-Compose-Reinforce (GCR; Li et al., 2025) as an end-to-end baseline, since it trains single-task policies and learns its value functions offline from labeled trajectories, which are unavailable for our environments. However, P^{2}BRS without persistence gating (Appendix N.3.2) composes robustness metrics with the same $\min/\max$ operators that GCR applies at the formula level, and thus isolates the effect of this composition scheme alone in our setting.

N.3.1 Comparison to Automaton-Based PBRS

We employ the approach in Section 4.4 of Icarte et al. (2022) as a representative example for automaton-based PBRS. The idea is to derive the potential function Φ from the LDBA $\mathcal{B}^\varphi$ alone, by treating $\mathcal{B}^\varphi$ itself as a standalone MDP $\mathcal{M}_{\mathcal{B}^\varphi}$: its states are the automaton states $q^\varphi \in \mathcal{Q}^\varphi$, its actions are the automaton's assignments and ε-transitions $\sigma^\varphi \in \Sigma^\varphi \cup \mathcal{E}^\varphi$, transitions follow the transition function δ^φ, the reward is the task reward emitted on each transition (here, $+1$ upon visiting an accepting state $q^\varphi \in \mathcal{F}^\varphi$ and 0 otherwise), and the discount factor is the γ of the environment MDP $\mathcal{M}$. Value iteration on this abstract MDP yields optimal values $V^*_{\mathcal{B}^\varphi}(q^\varphi)$, corresponding to the return achieved by an idealized agent able to trigger any desired automaton transition at every environment step. The potential function is then defined as $\Phi(s, q^\varphi) = -V^*_{\mathcal{B}^\varphi}(q^\varphi)$, and is independent of environment state s. The potential function is overridden as $\Phi(s, q^\varphi) = 0$ at terminal states to ensure PBRS invariance; Icarte et al. (2022) note that this leads to a positive reward spike on episode termination regardless of outcome (so long as the task reward is always non-negative).

StructLTL trains directly on reach-avoid sequences, which are equivalent to chains of automaton states. In this setting, value iteration admits the closed form $V^*_{\mathcal{B}^\varphi}(q^\varphi) = \gamma^{d(q^\varphi)-1}$, with $d(q^\varphi)$ representing the number of transitions separating q^φ from acceptance, and thus $\Phi(s, q^\varphi) = -\gamma^{d(q^\varphi)-1}$. The resulting shaped reward is readily characterized: a step that advances the automaton receives no additional reward (the potential decrease is exactly absorbed by discounting), while a step that self-loops at distance $d(q^\varphi)$ receives the increment $(1-\gamma)\gamma^{d(q^\varphi)-1}$. Progress is thus encouraged not through one-off bonuses on advancing transitions, but through a higher per-step annuity at automaton states closer to acceptance. However, Φ under automaton-based PBRS is by construction constant in the environment state s; its gradient within any given automaton state is identically zero. Automaton-based PBRS therefore fundamentally cannot provide guidance *within* a subgoal, only across subgoal boundaries.

Figure 10a presents results in *X2DronesEnv*. Interestingly, Plato with automaton-based PBRS performs markedly *worse* during early training compared to even unguided Plato, with low success rate paired with a low average episode length; this represents the agent learning to crash the drones into the world bounds and terminate the episode early. We attribute this behavior to the positive reward spikes upon termination conceded by Icarte et al. (2022); while an *optimal* agent would be unbiased from these spikes, an early agent is not. To an untrained agent, the reward spike upon crashing represents a one-step, zero-credit-assignment positive reward at the world bounds. Only once the critic has internalized Φ is the advantage from the positive spike canceled and does the perverse incentive evaporate; this incurs a $10\,\mathrm{M}$-step training lag before catching up to unguided Plato. While P^{2}BRS can generate similar reward spikes upon termination in principle, the dense guidance that it provides toward subgoal satisfaction clearly overpowers this perverse incentive in practice.

During late training, we see slight gains in performance with automaton-based PBRS compared to unguided Plato, but ultimately no qualitative difference in behavior, unlike P^2BRS which demonstrates a substantial learning speedup to convergence.

Figure 10b presents results in *FrankaCubesEnv*, in which Plato with automaton-based PBRS flatlines alongside unguided Plato. It is in this environment that the fundamental inadequacy of automaton-based PBRS is most clear; its high-level guidance has no effect if the agent is incapable of satisfying individual subgoals in the first place. This emphasizes the necessity for dense guidance towards subgoal satisfaction as is provided by P^2BRS, with which Plato converges to a strong success rate.

N.3.2 Impact of Persistence Gating

When tasked with a clause such as $\mathsf{has}_i \wedge \mathsf{pos}_j$, the agent must learn to satisfy both propositions simultaneously. However, moving to satisfy one proposition often means moving away from satisfying the other, in this case because the cube and the target position are in different locations. Thus, the agent must learn *which* proposition to satisfy first (picking up the cube), and then persistently satisfy that proposition while moving to satisfy the other (carrying it to the target position). While this *implicit* temporal ordering is induced by the environment dynamics, the clause itself provides no *explicit* structural guidance for it, meaning that the potential function produced by P^2BRS without persistence gating would guide the agent roughly towards the midpoint of the two locations (due to the $\min$ operator). The agent must then rely on exploration noise to discover how to satisfy the clause, which is virtually impossible for this environment. This can be resolved by the simple labeling of has as *persistent* and pos (and rot) as *non-persistent*, which here represents the common-sense domain knowledge that in order to take the cube to a target position, you have to pick it up first. The agent's *goal* remains unchanged, but the *guidance* from P^2BRS prioritizes picking up the cube. Note that the ungated composition coincides with the plain min/max operators of Li et al. (2025), with our robustness metrics in place of GCR's learned value functions, so the results below also indicate that such operators alone are insufficient for heterogeneous conjunctions in our setting.

Figure 10b presents results in *FrankaCubesEnv* without persistence gating. We see that performance quickly flatlines at around 10% success rate; the agent scores highly on φ_{17} (a sequence of position and rotation targets) but fails on all the other tasks, which involve variants of picking up cubes and carrying them to a target location or orientation. Thus, persistence gating can be essential to learn heterogeneous conjunctions with implicit temporal ordering induced by the environment dynamics.

We omit performance on *X2DronesEnv* since persistence gating has no effect, because there is only one predicate (pos); both drones can fly towards different target positions simultaneously.

N.3.3 Impact of Adaptive Robustness Scale Calibration

Recall from Section 6.1.1 that the robustness scale $\kappa_{\mathsf{f}}\tilde{g}_{\mathsf{f}}$ controls the breadth of guidance from the decision boundary; $\tilde{g}_{\mathsf{f}}$ is calibrated at the start of training and κ_{f} is calibrated online to track the agent's typical distance from it. In this ablation, we disable this calibration by freezing $\kappa_{\mathsf{f}}\tilde{g}_{\mathsf{f}} = 1$.

A frozen unit robustness scale commits every predicate, for all of training, to the sensitivity regime that its raw metric g_{f} happens to occupy at its native scale: a metric whose typical magnitudes far exceed 1 saturates ρ_{f} towards ± 1 with near-flat gradients away from a thin shell around the decision boundary, while a metric with typical magnitudes far below 1 compresses ρ_{f} towards zero, providing weak preference between states. Whether the freeze is harmful therefore depends on how far the calibrated robustness scale sits from 1, and on how much it moves as the trajectory distribution shifts, which Figure 4 reveals directly.

In *X2DronesEnv*, the calibrated scale for pos decays steadily over the first fifth of training before settling at around 0.8, close to the frozen unit scale (dotted red line in Figure 4). A unit freeze is thus coincidentally near-calibrated for this environment, and correspondingly, Figure 10a shows performance with and without calibration to be indistinguishable. We caution against reading this as evidence that calibration is unnecessary: the agreement is an artifact of the calibrated robustness scale happening to coincide with unity.

FrankaCubesEnv is the informative case. Figure 4 shows the calibrated scales separating by predicate, with rot settling at around 1.8 and pos at around 0.9, both after a steady decay over the first half of training, and has at around 0.1, roughly an order of magnitude below the frozen unit scale.

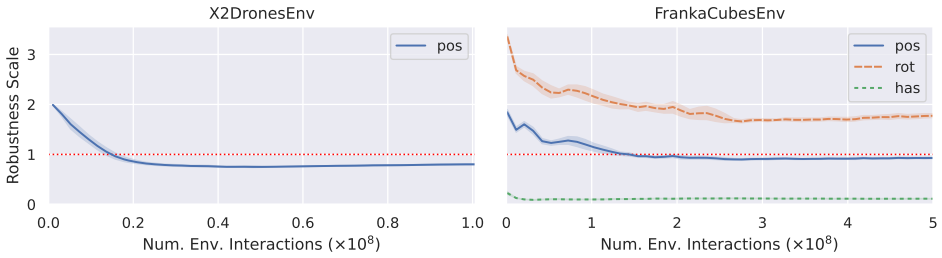


Figure 4: Robustness scales during training for Plato + P^{2}BRS. The dotted red line represents the frozen unit scale under the ablation of no adaptive robustness scale calibration.

The decay of the pos and rot scales agrees with our claim that online calibration behaves as the automatic curriculum described in Section 6.1.1, providing broad guidance while early trajectories remain far from the decision boundaries and tightening it as they concentrate; a frozen scale forgoes this tracking outright. The unit freeze also mis-scales the predicate that matters most: with typical magnitudes of g_{has} roughly an order of magnitude below unity, the uncalibrated ρ_{has} is compressed towards zero, weakening guidance in the gated first phase that persistence gating prioritizes, for the subgoal (grasping the correct cube) that is hardest to discover through exploration. It additionally distorts the relative weighting of rot against pos within the non-persistent phase, where the two are aggregated by a hard $\min$. Consistent with this, Figure 10b shows the uncalibrated variant converging markedly more slowly and plateauing at a lower success rate than full P^{2}BRS, with correspondingly longer episode lengths.

Thus, Figure 4 makes visible the two roles of adaptive calibration: normalizing heterogeneous predicates onto a common scale, and tracking the shifting trajectory distribution as an automatic curriculum. The results from *X2DronesEnv* demonstrate that the freeze is tolerable when the calibrated scale happens to coincide with unity, but when raw metric values are distributed far from unity, as for has and rot in *FrankaCubesEnv*, calibration becomes essential. Meanwhile, its cost is negligible, requiring only quantile tracking over a sliding window of already-computed metric values.

N.3.4 Impact of Smooth Combinators

The hard $\min/\max$ aggregations described in Section 6.1.2 propagate gradient to only a single operand — the arg-min (or arg-max) — leaving every other unsatisfied conjunct (or disjunct) with zero gradient. For a conjunctive subgoal such as $\mathsf{pos}_i \wedge \mathsf{pos}_j$ in *X2DronesEnv*, this starves all but the worst-satisfied conjunct of guidance at any given time step, even when the agent can make simultaneous progress on both. For disjunctions, it greedily starves all but the best-satisfied disjunct.

Inspired by smooth $\min/\max$ reformulations for trajectory optimization from the STL literature (Han et al., 2026), we design *smooth product t-norm combinators*. Assuming bounded robustness metrics $\rho_{\beta_j}(s) \in [-1, 1]$, we soften each aggregation analytically as follows:

$$\beta = \bigwedge_{j=1}^{m} \beta_j \iff \rho_\beta(s) = \begin{cases} \prod_{j=1}^{m} \big(1 - \max\{-\rho_{\beta_j}(s), 0\}\big) - 1 & \text{if } \min_{j\in 1:m} \rho_{\beta_j}(s) < 0, \\ \min_{j\in 1:m} \rho_{\beta_j}(s) & \text{else,} \end{cases}$$

$$\beta = \bigvee_{j=1}^{m} \beta_j \iff \rho_\beta(s) = \begin{cases} -\prod_{j=1}^{m} \big(-\rho_{\beta_j}(s)\big) & \text{if } \max_{j\in 1:m} \rho_{\beta_j}(s) < 0, \\ \max_{j\in 1:m} \rho_{\beta_j}(s) & \text{else.} \end{cases}$$

The conjunction yields a gradient term $\partial\rho_\beta/\partial\rho_{\beta_k} = \mathbb{1}(\rho_{\beta_k}(s) < 0) \prod_{j\neq k}(1 - \max\{-\rho_{\beta_j}(s), 0\})$ that pulls each unsatisfied conjunct ρ_{β_k} towards satisfaction, not just the furthest; the gradient magnitude grows the closer the other conjuncts are to satisfaction, which is a smooth approximation of the hard $\min$ aggregation gradient. Meanwhile, the disjunction yields a gradient term $\partial\rho_\beta/\partial\rho_{\beta_k} = \prod_{j\neq k}(-\rho_{\beta_j}(s))$ within the all-negative orthant that pulls each disjunct ρ_{β_k} towards

satisfaction, not just the closest; the gradient magnitude grows the further the other disjuncts are from satisfaction, which is a smooth approximation of the hard $\max$ aggregation gradient.

Note that smoothing is applied only while β is unsatisfied, and the satisfied regions retain the hard $\min/\max$ aggregation. The smoothed combinators are sign-exact to their hard counterparts, so the labeling function $L_\beta(s) = \mathcal{H}\big(\rho_\beta(s)\big)$ is preserved, and their outputs are bounded to $[-1, 1]$. Importantly, they also reduce to identity in the case of a single conjunct or disjunct.

Figure 10 presents results in *X2DronesEnv* and *FrankaCubesEnv* with P^{2}BRS augmented with smooth combinators; in both environments, performance gains relative to standard P^{2}BRS are inconsequential. This is somewhat to be expected for *FrankaCubesEnv*, wherein persistence gating overrides the smooth $\min$ combinator when has is present in a clause (which is true for the majority of clauses). However, the trend persists in *X2DronesEnv*, which represents the ideal use case with two drones simultaneously moving to different target positions.

The reason boils down to multi-step credit assignment. Since potential-based shaping leaves the optimal policy unchanged for any potential function, the smoothness of the combinator can influence learning dynamics only, and the natural concern is that the hard $\min$ combinator provides no gradient for any conjunct other than the current minimizer. This flatness is an *instantaneous* property, however, and the *learning* signal is not instantaneous. Over a trajectory segment of many steps the minimizer is likely to alternate between conjuncts, especially during early learning when policies are highly stochastic, and improvement itself drives the minimum up to meet the next conjunct. Thus, the gradient contribution typically alternates between conjuncts over a segment. The learning signal aggregates these contributions: each GAE advantage estimate sums TD residuals, and with them the shaping increments, over an effective horizon of many steps, and the critic's bootstrapped values propagate them further. The update an action receives is therefore a smoothed temporal average of per-conjunct gradients, weighted by the time each conjunct spends as the minimizer, much as the smooth combinator weights conjunct gradients by their robustness gaps at a single step.

For segments with a leading conjunct whose robustness stays well clear of the minimum (and therefore never alternates as the minimizer), the smooth combinator's sensitivity to the leading conjunct is low, decaying toward 0 as the robustness gap increases; thus, the benefit of analytical smoothing would be inconsequential whenever temporal smoothing from alternation does not already apply. This carries over to disjunctions; near-tied disjuncts are temporally smoothed by alternation of the maximizer, as above, and when one disjunct leads, the analytical smooth maximum's weights on the trailing disjuncts decay toward 0.

Multi-step advantage estimation thus already performs the gradient smoothing that smooth combinators provide analytically, making the latter redundant. Accordingly, we omit smooth combinators from the proposed P^{2}BRS design in the main paper, since we cannot justify their usefulness theoretically or empirically.

O LIMITATIONS

Specification and vocabulary. While PlatoLTL generalizes parametrically across instances of known predicates, it cannot generalize to entirely new ones, and thus requires all predicates in the evaluation set to be represented within the training curriculum. Note that prior LTL-guided RL approaches share an analogous limitation, requiring all propositions to be known a priori; PlatoLTL's predicate-based formulation is strictly more general. Moreover, a parameterization always exists, whether trivially (each proposition as its own zero-parameter predicate) or via a predicate parameterized by satisfying sets $\mathcal{G} \subseteq \mathcal{S}$; the gains of the Plato embedding module instead require parameters that are both explicitly specifiable and structurally related to the labeling function (cf. Appendix N.1), where the former construction lacks the structure and the latter the specifiability. The choice of specification granularity behaves similarly: under the degenerate specification F done, the automaton exposes no subgoals, and PlatoLTL reduces to sparse-reward RL for the monolithic task, whereas a granular specification over simpler predicates exposes subgoals as individually shaped milestones, at the cost of prescribing one particular means of achieving the task (cf. Appendix E). The specification and vocabulary are thus modeling choices: we hypothesize that a small number of well-chosen predicates suffices to capture a meaningful range of tasks, as small libraries of param-

eterized action primitives do for manipulation (Hausknecht & Stone, 2016; Dalal et al., 2021), and we discuss automating these choices via predicate discovery as future work in Appendix P.

Robustness metrics. The simple pose-based priors used throughout our environments assume access to the poses of task-relevant entities; these are directly available in simulation, and mature methods exist for estimating them from raw sensing (Xiang et al., 2018; Labbé et al., 2022; Wen et al., 2024). Since shaping with any bounded potential leaves the optimal policy unchanged, estimation error degrades only the quality of guidance; this is in contrast to the labeling function, whose accuracy directly determines task satisfaction semantics. A separate challenge is the construction of informative metrics. A valid metric always exists, as any metric on $\mathcal{S}$ induces one via the signed distance to a predicate's satisfying set; however, for semantically complex predicates (e.g., that an assembly is complete, or that a deformable object is folded), the satisfying set is specified only implicitly, and a hand-specified surrogate must instead compose multiple geometric quantities in nontrivial and potentially misleading ways. The challenge is thus one of construction cost rather than of existence. Under weak or trivial metrics, P^{2}BRS degrades gracefully to unguided exploration for the affected predicate; however, as our results demonstrate, unguided exploration can be the bottleneck for learning to satisfy the hardest predicates. We further note that while P^{2}BRS guarantees invariance of the optimal policy, it does not guarantee that shaping helps: an actively misleading surrogate could in principle slow learning relative to no shaping at all, and the practical burden is thus to supply metrics that are coarsely correct (or else use the trivial one).

Persistence annotations. Persistence annotations (Section 6.1.2) are likewise coarse, manually supplied domain knowledge, and affect only the guidance and not the optimal policy; however, given that persistence gating can be essential for heterogeneous conjunctions (Appendix N.3.2), a poorly annotated vocabulary could forfeit much of the benefit of P^{2}BRS. A predicate's persistence may also in principle be context-dependent rather than fixed; inferring persistence automatically, e.g., from demonstration trajectories, is an interesting direction for removing this annotation burden.

P Directions for Future Work

Uncertain Labeling Functions. Like most work in LTL-guided RL, we assume access to a ground-truth labeling function; in real-world applications, labels would instead be estimated by perception and are therefore uncertain. A natural direction for future work is making PlatoLTL robust in this regime. Prior work handles uncertain labels via joint action and query policies (Hatanaka et al., 2023) or beliefs over automaton states (Li et al., 2024); such approaches operate at the labeling and automaton layer, and are thus, in principle, compatible with PlatoLTL, whose contributions concern the goal embedding and reward shaping. An open question is how the potential function of P^{2}BRS should be evaluated when the automaton state itself becomes a belief, e.g., in expectation.

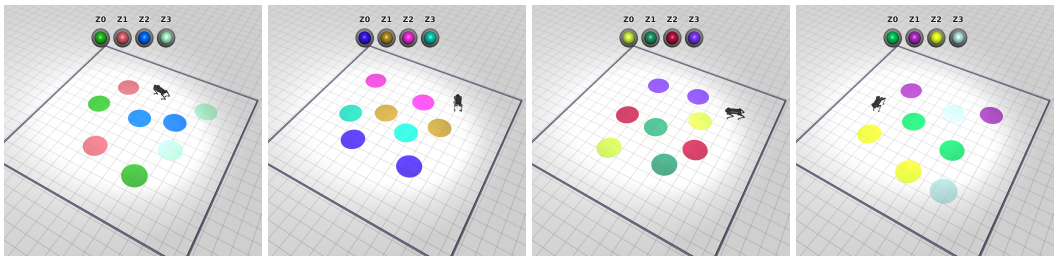
Learned Labeling Functions and Robustness Metrics. A complementary direction is to learn the labeling functions from data, e.g., as per-predicate success classifiers $\hat{L}_{\mathsf{f}}(s, x_{\mathsf{f}})$ trained on labeled outcomes (Fu et al., 2018; Eysenbach et al., 2021; Du et al., 2023). Notably, this would unify the labeling function and the robustness metric into a single learned artifact: thresholding the classifier score recovers the label, while the score itself serves as a raw metric satisfying sign-consistency by construction, offering a route to metrics for the semantically complex predicates of Appendix O. Denser guidance could instead be derived from goal-conditioned value functions or temporal distances, pre-trained on passive video or demonstrations (Ma et al., 2023a;b) or learned per proposition via offline RL from labeled trajectories (Li et al., 2025); such metrics need not agree in sign with the labeling function, since the construction of Section 6.1 repairs any raw metric $\hat{g}_{\mathsf{f}}$ into a valid g_{f} given L_{f}, and the adaptive calibration of Section 6.1.1 consumes metrics of arbitrary scale without manual tuning. Under our predicate formulation, such models would instead be conditioned on the parameters, so that their number scales with $|Pred|$ rather than with the number of propositions. Learned metrics do, however, require training data with adequate state coverage, which is unavailable for the environments studied here; our simple priors avoid this requirement.

Automatic Predicate Discovery. A promising direction for relaxing the assumption that predicates are known a priori (Appendix O) is automatic predicate discovery: recent work explores the discovery of grounded predicates from raw observations or natural language (Asai, 2019; Pang et al., 2023; Gupta et al., 2025). Future work could leverage these methods to identify useful predicates and parameterizations in a pre-training phase, which are then consumed downstream by PlatoLTL.

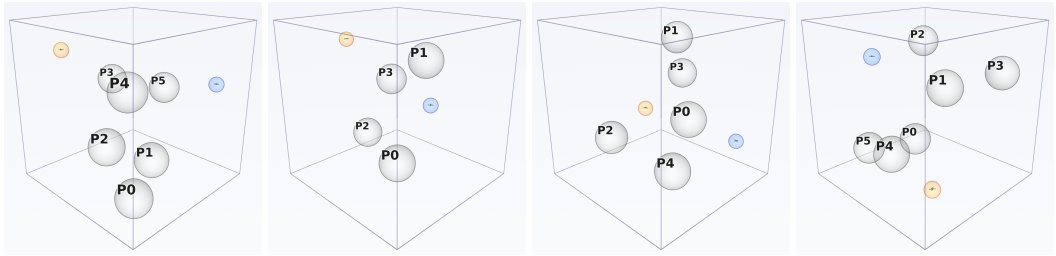
LTL from Natural Language. Another natural direction for future work is grounding PlatoLTL in natural language (NL) interfaces. Like all LTL-guided RL approaches, PlatoLTL and its backbones require tasks to be specified in LTL syntax, which trades accessibility for precision and verifiability. This is in contrast to approaches such as Vision-Language-Action models (VLAs; Zitkovich et al., 2023; Kim et al., 2024; Black et al., 2025), which condition agents directly on NL.

Formal specification languages such as LTL come with the benefit of being unambiguous and verifiable, allowing users to precisely define the problem and rigorously monitor task progress. LTL is well-established in formal verification, where it is routinely used to specify properties of safety-critical systems such as nuclear power plants and aircraft control systems (Baier & Katoen, 2008); indeed, safety-critical applications often require LTL specifications for this reason. Our *Fallout-World* environment is one such example, representing inspection of nuclear sites. Furthermore, unlike VLAs, LTL-guided approaches like PlatoLTL require comparatively small, inexpensive networks while still generalizing across tasks with high specification complexity.

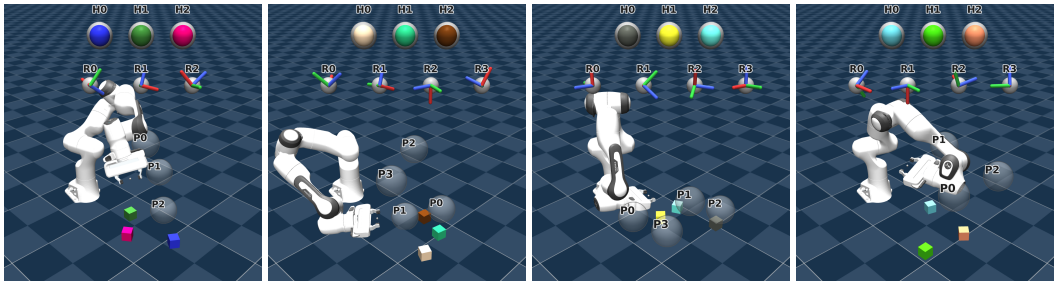
Recent work demonstrates grounded automatic translation from NL to LTL (Fuggitti & Chakraborti, 2023), as well as zero-shot autoformalization of reward machines from NL reward descriptions (Li et al., 2025). This suggests that PlatoLTL could, in principle, inherit the accessibility of NL interfaces while retaining the interpretability and verifiability benefits of LTL specifications.



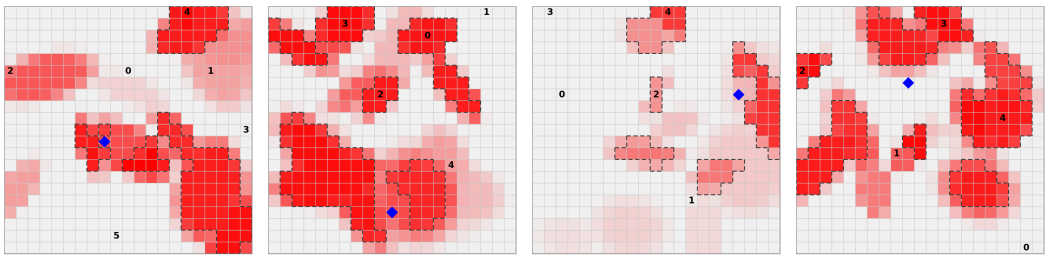
(a) *Go1RGBZoneEnv*. The legend marks the predicate instance for each color.



(b) *X2DronesEnv*. The blue and orange circles highlight the two drones.



(c) *FrankaCubesEnv*. The legend marks the predicate instance for each color and orientation.



(d) *FalloutWorld*. The blue diamond marks the agent and the dashed borders represent the rad threshold.

Figure 5: Visualizations of example environment reset configurations, including predicate instances.

Table 4: Template LTL specifications used for evaluation. The notation f_i represents the i-th instance of the predicate f.

(a) Finite-horizon specifications φ

	φ	Specification	Summary
Go1RGBZoneEnv	φ_1	$\mathsf{F}(\mathsf{at}_0 \wedge (\neg \mathsf{at}_1 \,\mathsf{U}\, \mathsf{at}_2)) \wedge \mathsf{F}\, \mathsf{at}_3$	Sequential reach with temporary obstacle avoidance
	φ_2	$\mathsf{F}\, \mathsf{at}_0 \wedge (\neg \mathsf{at}_0 \,\mathsf{U}\, (\mathsf{at}_1 \wedge \mathsf{F}\, \mathsf{at}_2))$	Guarded sequential reach
	φ_3	$\mathsf{F}(\mathsf{at}_0 \vee \mathsf{at}_1) \wedge \mathsf{F}\, \mathsf{at}_2 \wedge \mathsf{F}\, \mathsf{at}_3$	Unordered reach with choice
	φ_4	$\neg(\mathsf{at}_0 \vee \mathsf{at}_1) \,\mathsf{U}\, (\mathsf{at}_2 \wedge \mathsf{F}\, \mathsf{at}_3)$	Sequential reach with global obstacle avoidance
	φ_5	$\neg \mathsf{at}_0 \,\mathsf{U}\, ((\mathsf{at}_1 \vee \mathsf{at}_2) \wedge (\neg \mathsf{at}_0 \,\mathsf{U}\, \mathsf{at}_3))$	Sequential reach with choice and obstacle avoidance
	φ_6	$((\mathsf{at}_0 \vee \mathsf{at}_1) \Rightarrow (\neg \mathsf{at}_2 \,\mathsf{U}\, \mathsf{at}_3)) \,\mathsf{U}\, \mathsf{at}_2$	Reach with post-hazard recovery
	φ_7	$\mathsf{F}(\mathsf{at}_0 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_1 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_2 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_3 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_2 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_0 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_3 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_1 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_3 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_1 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{at}_2 \wedge \mathsf{X}\,\mathsf{F}\, \mathsf{at}_0))))))))))$	12-step reach
	φ_8	$\neg \mathsf{at}_0 \,\mathsf{U}\, (\mathsf{at}_1 \wedge (\neg \mathsf{at}_2 \,\mathsf{U}\, (\mathsf{at}_0 \wedge (\neg \mathsf{at}_3 \,\mathsf{U}\, (\mathsf{at}_2 \wedge (\neg \mathsf{at}_0 \,\mathsf{U}\, (\mathsf{at}_3 \wedge (\neg \mathsf{at}_2 \,\mathsf{U}\, (\mathsf{at}_1 \wedge (\neg \mathsf{at}_0 \,\mathsf{U}\, (\mathsf{at}_2 \wedge (\neg \mathsf{at}_3 \,\mathsf{U}\, (\mathsf{at}_0 \wedge (\neg \mathsf{at}_1 \,\mathsf{U}\, \mathsf{at}_3))))))))))))))$	8-step reach-avoid
X2DronesEnv	φ_9	$\neg \mathsf{pos}_0 \,\mathsf{U}\, (\mathsf{pos}_1 \vee (\mathsf{pos}_2 \wedge \mathsf{pos}_3))$	Reach with choice and global obstacle avoidance
	φ_{10}	$\mathsf{F}\, \mathsf{pos}_0 \wedge (\neg(\mathsf{pos}_1 \vee \mathsf{pos}_2) \,\mathsf{U}\, \mathsf{pos}_3)$	Reach with temporary obstacle avoidance
	φ_{11}	$((\mathsf{pos}_0 \vee \mathsf{pos}_1 \vee \mathsf{pos}_2) \Rightarrow \mathsf{F}(\mathsf{pos}_3 \wedge \mathsf{pos}_4)) \,\mathsf{U}\, (\mathsf{pos}_2 \wedge \mathsf{pos}_3)$	Simultaneous reach with post-hazard recovery
	φ_{12}	$\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}((\mathsf{pos}_1 \wedge \mathsf{pos}_2) \wedge \mathsf{F}((\mathsf{pos}_2 \wedge \mathsf{pos}_3) \wedge (\neg(\mathsf{pos}_0 \vee \mathsf{pos}_1) \,\mathsf{U}\, (\mathsf{pos}_3 \wedge \mathsf{pos}_4)))))$	Mixed sequence including simultaneous reach
	φ_{13}	$\neg(\mathsf{pos}_0 \vee \mathsf{pos}_1 \vee (\mathsf{pos}_2 \wedge \mathsf{pos}_3)) \,\mathsf{U}\, ((\mathsf{pos}_3 \wedge \mathsf{pos}_4) \wedge \mathsf{F}(\mathsf{pos}_4 \wedge \mathsf{pos}_5))$	Simultaneous reach with global obstacle avoidance
	φ_{14}	$\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}(\mathsf{pos}_1 \wedge \mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{F}(\mathsf{pos}_3 \wedge \mathsf{F}(\mathsf{pos}_4 \wedge \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{F}\, \mathsf{pos}_4)))))))$	8-step reach
	φ_{15}	$(\neg \mathsf{pos}_0 \,\mathsf{U}\, (\mathsf{pos}_1 \wedge \mathsf{F}\, \mathsf{pos}_2)) \vee (\neg \mathsf{pos}_0 \,\mathsf{U}\, (\mathsf{pos}_3 \wedge \mathsf{F}\, \mathsf{pos}_4))$	Choice of two reach-avoid paths
	φ_{16}	$(\neg(\mathsf{pos}_0 \vee \mathsf{pos}_1) \,\mathsf{U}\, \mathsf{pos}_2) \wedge \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{pos}_1)$	Guarded simultaneous reach
FrankaCubesEnv	φ_{17}	$\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}(\mathsf{rot}_0 \wedge \mathsf{F}(\mathsf{pos}_1 \wedge \mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{F}(\mathsf{rot}_1 \wedge \mathsf{F}(\mathsf{pos}_3 \wedge \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}(\mathsf{rot}_0 \wedge \mathsf{F}\, \mathsf{pos}_2))))))))$	9-step sequence of position and orientation targets
	φ_{18}	$\neg \mathsf{pos}_0 \,\mathsf{U}\, (\mathsf{pos}_i \wedge \mathsf{has}_0)$	Pick-and-carry with global obstacle avoidance
	φ_{19}	$\mathsf{F}(\mathsf{has}_0 \wedge \mathsf{F}((\mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge \mathsf{F}(\mathsf{has}_0 \wedge \mathsf{pos}_1 \wedge \mathsf{rot}_0)))$	Pick-and-carry to a target pose
	φ_{20}	$\mathsf{F}((\mathsf{has}_0 \wedge \mathsf{rot}_0) \wedge \mathsf{F}((\mathsf{has}_0 \wedge \mathsf{rot}_1) \wedge \mathsf{F}(\mathsf{has}_0 \wedge (\mathsf{pos}_0 \vee \mathsf{pos}_1))))$	Oriented carry with choice of position target
	φ_{21}	$\neg \mathsf{pos}_0 \,\mathsf{U}\, (\mathsf{has}_0 \wedge (\neg \mathsf{pos}_1 \,\mathsf{U}\, (\mathsf{has}_0 \wedge \mathsf{pos}_2)))$	Pick-and-carry with phased obstacle avoidance
	φ_{22}	$\mathsf{F}((\mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge (\mathsf{pos}_0 \,\mathsf{U}\, (\neg \mathsf{has}_0 \wedge \mathsf{pos}_0))) \wedge \mathsf{F}((\mathsf{has}_1 \wedge \mathsf{pos}_1) \wedge (\mathsf{pos}_1 \,\mathsf{U}\, (\neg \mathsf{has}_1 \wedge \mathsf{pos}_1)))$	Unordered 2-object pick-and-place
	φ_{23}	$\mathsf{F}((\mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge (\mathsf{pos}_0 \,\mathsf{U}\, ((\neg \mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge \mathsf{F}((\mathsf{has}_0 \wedge \mathsf{pos}_1) \wedge (\mathsf{pos}_1 \,\mathsf{U}\, (\neg \mathsf{has}_0 \wedge \mathsf{pos}_1))))))$	Pick-and-place with relocation
	φ_{24}	$\mathsf{F}((\mathsf{has}_0 \wedge (\mathsf{pos}_0 \vee \mathsf{pos}_1)) \wedge ((\mathsf{pos}_0 \vee \mathsf{pos}_1) \,\mathsf{U}\, (\neg \mathsf{has}_0 \wedge (\mathsf{pos}_0 \vee \mathsf{pos}_1))))$	Pick-and-place with choice of target position
FalloutWorld	φ_{25}	$\neg(\mathsf{rad}_0 \vee \mathsf{loc}_0) \,\mathsf{U}\, ((\mathsf{loc}_1 \wedge \neg \mathsf{rad}_0) \wedge \mathsf{F}\, \mathsf{loc}_2)$	Reach with temporary radiation avoidance
	φ_{26}	$\neg \mathsf{rad}_0 \,\mathsf{U}\, ((\mathsf{loc}_0 \vee \mathsf{loc}_1) \wedge (\neg \mathsf{rad}_0 \,\mathsf{U}\, (\mathsf{loc}_2 \wedge \neg \mathsf{rad}_0)))$	Reach with global radiation avoidance
	φ_{27}	$(\mathsf{rad}_0 \Rightarrow (\neg \mathsf{loc}_0 \,\mathsf{U}\, (\mathsf{loc}_1 \vee \mathsf{loc}_2))) \,\mathsf{U}\, (\mathsf{loc}_0 \wedge \neg \mathsf{rad}_0)$	Reach with post-hazard recovery (decontamination)
	φ_{28}	$(\neg \mathsf{rad}_0 \,\mathsf{U}\, (\mathsf{loc}_0 \wedge \neg \mathsf{rad}_0)) \wedge \mathsf{F}(\mathsf{rad}_0 \wedge \mathsf{F}\, \mathsf{loc}_1)$	Radiation-free reach followed by radiation probing
	φ_{29}	$\mathsf{F}\, \mathsf{loc}_0 \wedge \mathsf{F}\, \mathsf{loc}_1 \wedge (\neg \mathsf{rad}_0 \,\mathsf{U}\, (\mathsf{loc}_2 \wedge \neg \mathsf{rad}_0))$	Unordered rad-sensitive and rad-agnostic reach
	φ_{30}	$(\neg(\mathsf{loc}_0 \vee \mathsf{loc}_1) \,\mathsf{U}\, \mathsf{rad}_0) \wedge \mathsf{F}(\mathsf{loc}_0 \wedge \mathsf{F}\, \mathsf{loc}_1)$	Sequential reach guarded by radiation probing
	φ_{31}	$\mathsf{F}(\mathsf{loc}_0 \wedge \mathsf{F}(\mathsf{loc}_1 \wedge \mathsf{F}(\mathsf{loc}_2 \wedge \mathsf{F}(\mathsf{rad}_0 \wedge \mathsf{F}(\mathsf{loc}_3 \wedge \mathsf{F}(\mathsf{loc}_4 \wedge \mathsf{F}\, \mathsf{loc}_5))))))$	7-step reach including radiation probe
	φ_{32}	$((\neg \mathsf{rad}_0 \,\mathsf{U}\, (\mathsf{loc}_0 \wedge \neg \mathsf{rad}_0)) \wedge \mathsf{F}\, \mathsf{loc}_1) \vee ((\neg \mathsf{rad}_0 \,\mathsf{U}\, (\mathsf{loc}_2 \wedge \neg \mathsf{rad}_0)) \wedge \mathsf{F}\, \mathsf{loc}_3)$	Choice of two paths with temporary rad-avoidance

(b) Infinite-horizon specifications ψ

	ψ	Specification	Summary
Go1RGBZoneEnv	ψ_1	$\mathsf{F}\,\mathsf{G}\, \mathsf{at}_0$	Persistent reach
	ψ_2	$\mathsf{F}\,\mathsf{G}\, \mathsf{at}_0 \wedge \mathsf{F}(\mathsf{at}_1 \wedge \mathsf{F}\, \mathsf{at}_2)$	Persistent reach and sequential reach
	ψ_3	$\mathsf{F}\,\mathsf{G}\, \mathsf{at}_0 \wedge \mathsf{G}\, \neg \mathsf{at}_1$	Persistent reach with global obstacle avoidance
	ψ_4	$\mathsf{G}((\mathsf{at}_0 \vee \mathsf{at}_1) \Rightarrow \mathsf{F}\, \mathsf{at}_2) \wedge \mathsf{F}\, \mathsf{G}(\mathsf{at}_0 \vee \mathsf{at}_3)$	Persistent reach with post-hazard recovery
	ψ_5	$\mathsf{G}\,\mathsf{F}\, \mathsf{at}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{at}_1$	2-step cycle
	ψ_6	$\mathsf{G}\,\mathsf{F}\, \mathsf{at}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{at}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{at}_2 \wedge \mathsf{G}\, \neg \mathsf{at}_3$	Unordered 3-step cycle with global obstacle avoidance
X2DronesEnv	ψ_7	$\mathsf{G}\,\mathsf{F}\, \mathsf{pos}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_2 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_3 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_4 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_5$	Unordered 6-step cycle
	ψ_8	$\mathsf{G}\,\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{pos}_1) \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_2 \wedge \mathsf{G}\, \neg(\mathsf{pos}_3 \vee \mathsf{pos}_4)$	2-step mixed cycle with global obstacle avoidance
	ψ_9	$\mathsf{G}\,\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{pos}_1 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{X}\,\mathsf{F}(\mathsf{pos}_1 \wedge \mathsf{X}\,\mathsf{F}\, \mathsf{pos}_0)))) \wedge \mathsf{G}\, \neg \mathsf{pos}_3$	Ordered 4-step cycle with global obstacle avoidance
	ψ_{10}	$\mathsf{G}\,\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{pos}_1) \wedge \mathsf{G}\,\mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{pos}_3) \wedge \mathsf{F}(\mathsf{pos}_4 \wedge \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}\, \mathsf{pos}_2)) \wedge \mathsf{G}\, \neg \mathsf{pos}_5$	2-step cycle and sequential reach with global avoidance
	ψ_{11}	$\mathsf{G}\,\mathsf{F}\, \mathsf{pos}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_2 \wedge \mathsf{G}((\mathsf{pos}_3 \vee \mathsf{pos}_4) \Rightarrow \neg \mathsf{pos}_0 \,\mathsf{U}\, \mathsf{pos}_1)$	Unordered 3-step cycle with post-hazard recovery
	ψ_{12}	$\mathsf{G}\,\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{pos}_1) \wedge \mathsf{G}\,\mathsf{F}(\mathsf{pos}_2 \wedge \mathsf{pos}_3) \wedge \mathsf{G}\, \neg(\mathsf{pos}_0 \wedge \mathsf{pos}_2)$	2-step simultaneous reach cycle with a forbidden overlap
FrankaCubesEnv	ψ_{13}	$\mathsf{G}\,\mathsf{F}\, \mathsf{pos}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_2 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_3$	Unordered 4-step cycle of position targets
	ψ_{14}	$\mathsf{G}\,\mathsf{F}\, \mathsf{pos}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{pos}_2 \wedge \mathsf{G}\, \neg \mathsf{pos}_3 \wedge (\neg \mathsf{pos}_1 \,\mathsf{U}\, \mathsf{pos}_0) \wedge (\neg \mathsf{pos}_2 \,\mathsf{U}\, \mathsf{pos}_1)$	3-step cycle with ordered start and global avoidance
	ψ_{15}	$\mathsf{G}\,\mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}(\mathsf{rot}_0 \wedge \mathsf{F}(\mathsf{pos}_1 \wedge \mathsf{F}\, \mathsf{rot}_1))) \wedge \mathsf{G}\, \neg \mathsf{pos}_2$	Ordered 4-step mixed cycle with global obstacle avoidance
	ψ_{16}	$(\neg \mathsf{has}_0 \,\mathsf{U}\, (\mathsf{pos}_0 \vee \mathsf{pos}_1)) \wedge \mathsf{G}\, \neg(\mathsf{has}_0 \wedge (\mathsf{pos}_0 \vee \mathsf{pos}_1)) \wedge \mathsf{G}\,\mathsf{F}(\mathsf{has}_0 \wedge \mathsf{pos}_2) \wedge \mathsf{G}\,\mathsf{F}(\mathsf{has}_0 \wedge \mathsf{pos}_3)$	Pick-and-cycle with guarded pick and avoid while carrying
	ψ_{17}	$\mathsf{G}\,\mathsf{F}(\mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge \mathsf{G}\,\mathsf{F}(\mathsf{has}_1 \wedge \mathsf{pos}_1)$	Pick-and-carry cycle alternating between two cubes
	ψ_{18}	$\mathsf{G}\,\mathsf{F}\, \mathsf{has}_0 \wedge \mathsf{G}\,\mathsf{F}\, \neg \mathsf{has}_0 \wedge \mathsf{G}(\mathsf{has}_0 \Rightarrow \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}\, \mathsf{pos}_1))$	Recurrent pick-carry-release
FalloutWorld	ψ_{19}	$\mathsf{G}(\mathsf{rad}_0 \Rightarrow \mathsf{F}(\mathsf{loc}_0 \wedge \mathsf{F}\, \mathsf{loc}_1)) \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_2 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_3$	2-step cycle with post-hazard recovery (decontamination)
	ψ_{20}	$\mathsf{G}\,\mathsf{F}\, \mathsf{loc}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_2 \wedge \mathsf{G}\, \neg \mathsf{rad}_0$	Unordered 3-step cycle with global radiation avoidance
	ψ_{21}	$\mathsf{G}\,\mathsf{F}(\mathsf{loc}_0 \wedge \mathsf{F}(\mathsf{loc}_1 \wedge \mathsf{F}(\mathsf{loc}_2 \wedge \mathsf{F}\, \mathsf{loc}_3))) \wedge \mathsf{G}\, \neg \mathsf{rad}_0$	Ordered 4-step cycle with global radiation avoidance
	ψ_{22}	$\mathsf{G}\,\mathsf{F}\, \mathsf{loc}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_1 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_2 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_3 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_4 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_5$	Unordered 6-step cycle
	ψ_{23}	$\mathsf{G}\,\mathsf{F}(\mathsf{rad}_0 \wedge \mathsf{F}(\mathsf{loc}_0 \wedge \mathsf{F}(\neg \mathsf{rad}_0 \wedge \mathsf{F}\, \mathsf{loc}_1)))$	Ordered 4-step cycle including radiation cycling
	ψ_{24}	$\mathsf{G}\,\mathsf{F}\, \mathsf{rad}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_0 \wedge \mathsf{G}\,\mathsf{F}\, \mathsf{loc}_1 \wedge \mathsf{G}(\mathsf{rad}_0 \Rightarrow (\neg(\mathsf{loc}_0 \vee \mathsf{loc}_1) \,\mathsf{U}\, \mathsf{loc}_2))$	Unordered 3-step cycle including cyclical decontamination

Table 5: Details regarding the curriculum for sampling reach-avoid sequences during training.

(a) Curriculum stages. For each stage, we list the sequence type (reach-avoid/reach-stay), the probability $\mathrm{P}_{\mathrm{seq}}$ of sampling a sequence of that type, the number of steps n (or, for reach-stay tasks, the number of consecutive time steps for satisfying the "stay" sub-task), the sets of Boolean formulae from which each β_i^+ and β_i^- are sampled, and the probability $\mathrm{P}_{\mathrm{avd}}$ of sampling an active avoid-formula $\beta_i^- \neq \bot$ at each subgoal. Avoid-formulae that would overlap with the current or preceding reach-formula are discarded and resampled.

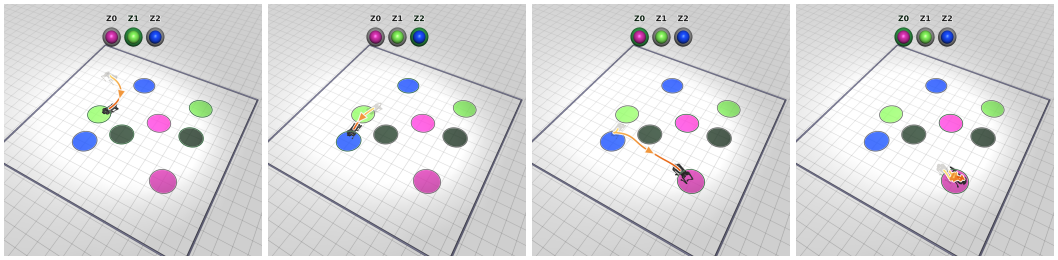
Stage	Type	$\mathrm{P}_{\mathrm{seq}}$	n	β_i^+	β_i^-	$\mathrm{P}_{\mathrm{avd}}$
1	Reach-Avoid	1.0	1	AP	$\bot$	0.0
2	Reach-Avoid	1.0	1	$\mathcal{V}_r$	$\bot$	0.0
3	Reach-Avoid	1.0	1	$\mathcal{V}_r$	$\mathcal{V}_n$	0.5
4	Reach-Avoid	1.0	1	$\mathcal{V}_r$	$\mathcal{V}_w$	0.7
5	Reach-Avoid	1.0	[1, 2]	$\mathcal{V}_r$	$\mathcal{V}_w$	0.7
6	Reach-Avoid	0.4	[1, 2]	$\mathcal{V}_r$	$\mathcal{V}_w$	0.7
	Reach-Stay	0.6	30	$\mathcal{V}_s$	$\mathcal{V}_n$	0.2
7	Reach-Avoid	0.8	[1, 2]	$\mathcal{V}_r$	$\mathcal{V}_w$	0.7
	Reach-Stay	0.2	60	$\mathcal{V}_s$	$\mathcal{V}_w$	0.5

(b) Sets of Boolean formulae from which β_i^+ and β_i^- are sampled during training. We define $\neg AP = \{\neg \mathsf{p} : \mathsf{p} \in AP\}$, $\bigwedge_k(S) = \{\beta_1 \wedge \cdots \wedge \beta_k : \beta_i \in S\}$, and $\bigvee_k(S)$ analogously. Note that the Boolean formulae must correspond to valid sets of assignments $\sigma \in \Sigma$ for each environment; if a formula contains an impossible clause (e.g., $\mathsf{at}_i \wedge \mathsf{at}_j$ conjunctions in *Go1RGBZoneEnv*), that formula is excluded.

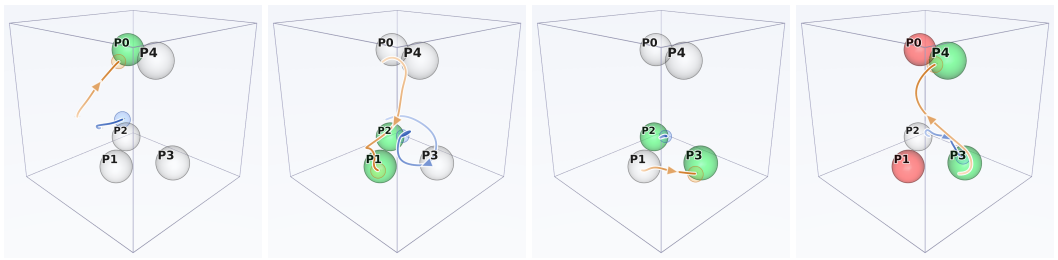
Set	Contents	Summary
Λ_k	$\bigwedge_k(AP)$	Clauses without negations
$\Lambda_k^{\pm}$	$\bigwedge_k(AP \cup \neg AP) \setminus \left(\bigwedge_k(AP) \cup \bigwedge_k(\neg AP)\right)$	Clauses with negations
D_p	$\bigcup_{k=2}^{4} \bigvee_k(AP)$	Disjunctions of propositions
D_m	$\bigvee_2(AP \cup \Lambda_2 \cup \Lambda_3 \cup \Lambda_2^{\pm}) \setminus \bigvee_2(AP)$	Disjunctions of clauses
$\mathcal{V}_r$	$AP \cup \Lambda_2 \cup \Lambda_3 \cup \Lambda_2^{\pm} \cup \Lambda_3^{\pm}$	"Reach" set
$\mathcal{V}_s$	$AP \cup \neg AP \cup \Lambda_2 \cup \Lambda_3 \cup \Lambda_2^{\pm}$	"Stay" set
$\mathcal{V}_n$	$AP \cup \Lambda_2$	"Narrow avoid" set
$\mathcal{V}_w$	$AP \cup \Lambda_2 \cup \mathrm{D}_p \cup \mathrm{D}_m$	"Wide avoid" set

(c) Template sets of propositions AP. For each predicate f we list the maximum number of predicate instances, as well as the probabilities $\mathrm{P}_{\mathrm{rch}}$ and $\mathrm{P}_{\mathrm{avd}}$ that an instance of that predicate is sampled when a proposition is sampled to build β_i^+ and β_i^-, respectively. While trained Plato and Discretize models admit an arbitrary number of predicate instances, Concatenate and SIAMS do not because the number of instances determines their model input sizes, and thus they require a maximum number of instances per predicate defined a priori.

Environment	f	Count	$\mathrm{P}_{\mathrm{rch}}$	$\mathrm{P}_{\mathrm{avd}}$
Go1RGBZoneEnv	at	4	1	1
X2DronesEnv	pos	6	1	1
FrankaCubesEnv	pos	4	$1/3$	$1/3$
	rot	4	$1/3$	$1/3$
	has	2	$1/3$	$1/3$
FalloutWorld	rad	1	$1/6$	$3/4$
	loc	6	$5/6$	$1/4$

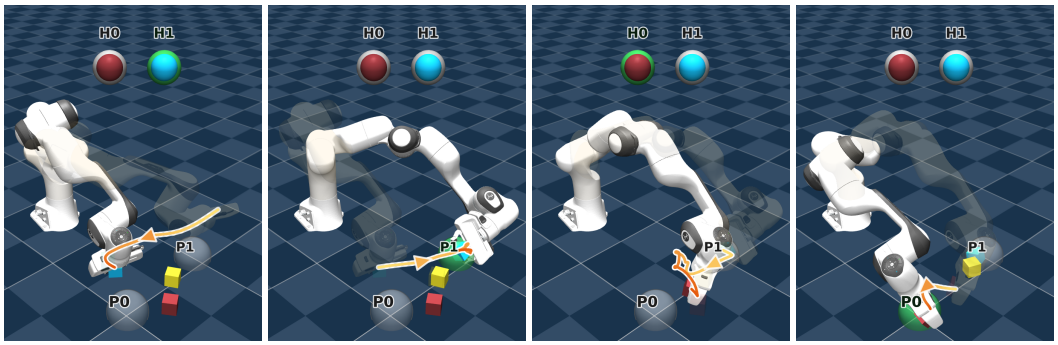


(a) *Go1RGBZoneEnv*; $\psi_2 = \mathsf{F}\,\mathsf{G}\,\mathsf{at}_0 \wedge \mathsf{F}(\mathsf{at}_1 \wedge \mathsf{F}\,\mathsf{at}_2)$.



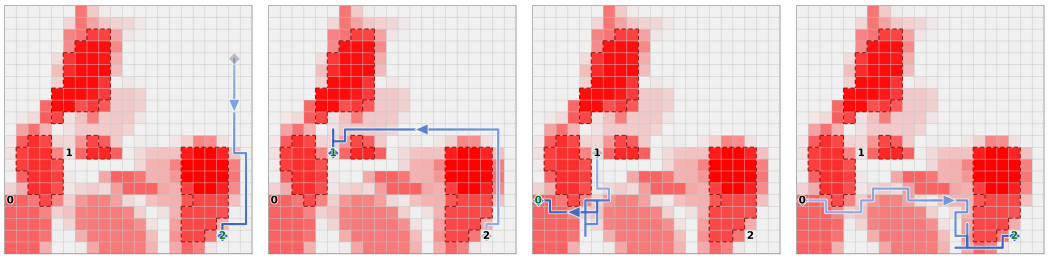
(b) *X2DronesEnv*;

$\varphi_{12} = \mathsf{F}(\mathsf{pos}_0 \wedge \mathsf{F}((\mathsf{pos}_1 \wedge \mathsf{pos}_2) \wedge \mathsf{F}((\mathsf{pos}_2 \wedge \mathsf{pos}_3) \wedge (\neg(\mathsf{pos}_0 \vee \mathsf{pos}_1)\,\mathsf{U}\,(\mathsf{pos}_3 \wedge \mathsf{pos}_4))))))$.



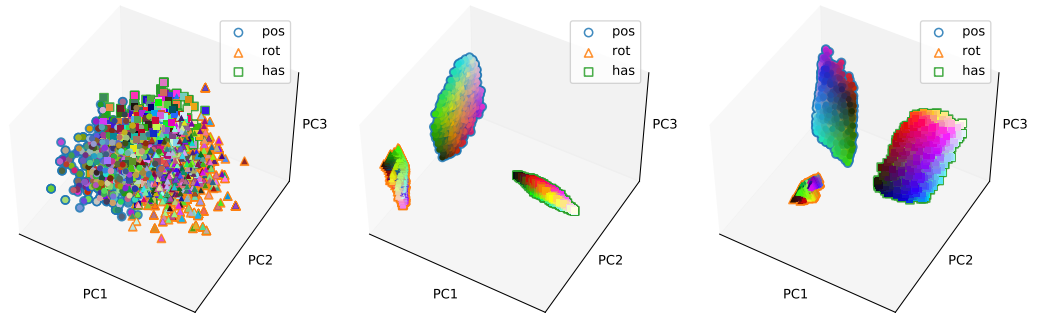
(c) *FrankaCubesEnv*;

$\varphi_{22} = \mathsf{F}((\mathsf{has}_0 \wedge \mathsf{pos}_0) \wedge (\mathsf{pos}_0\,\mathsf{U}\,(\neg\mathsf{has}_0 \wedge \mathsf{pos}_0))) \wedge \mathsf{F}((\mathsf{has}_1 \wedge \mathsf{pos}_1) \wedge (\mathsf{pos}_1\,\mathsf{U}\,(\neg\mathsf{has}_1 \wedge \mathsf{pos}_1)))$.

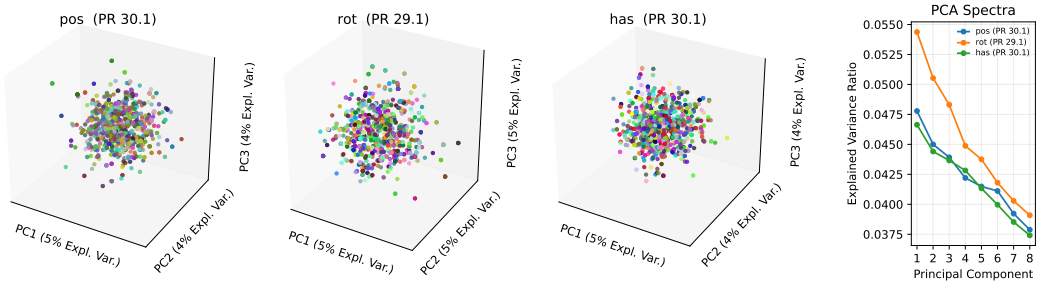


(d) *FalloutWorld*; $\psi_{20} = \mathsf{G}\,\mathsf{F}\,\mathsf{loc}_0 \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{loc}_1 \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{loc}_2 \wedge \mathsf{G}\,\neg\mathsf{rad}_0$.

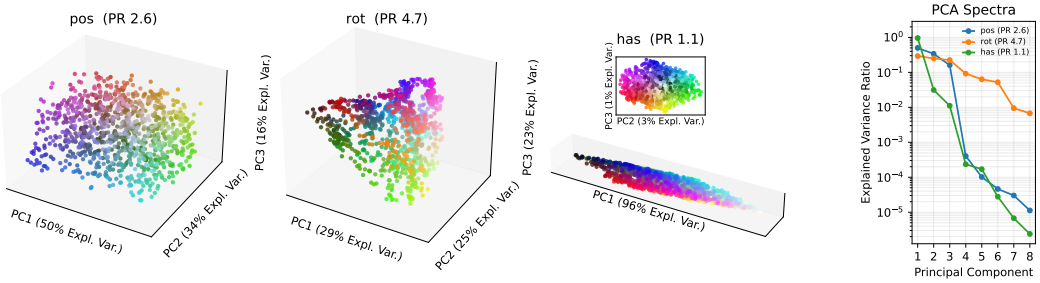
Figure 6: Example trajectories for Plato with P^{2}BRS, truncated to the first 4 subgoals (left to right).



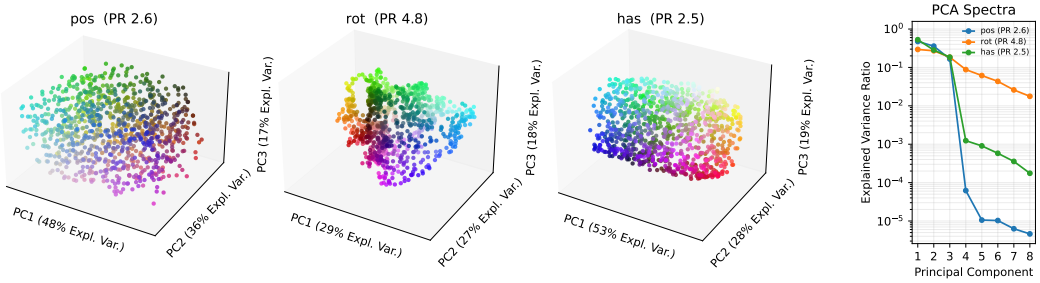
(a) Discretize (b) Plato (c) Plato + P^{2}BRS



(d) Per-predicate PCAs (Discretize), including explained variance ratios.

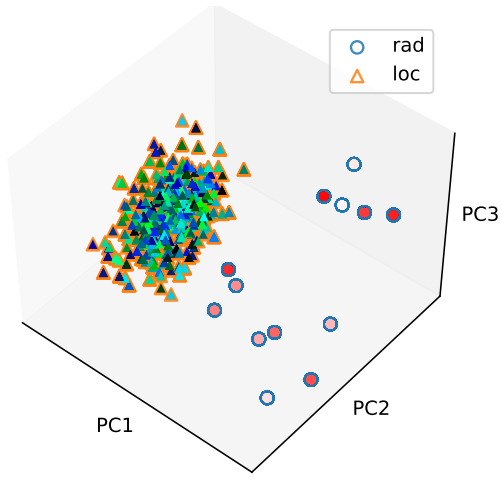


(e) Per-predicate PCAs (Plato), including explained variance ratios.

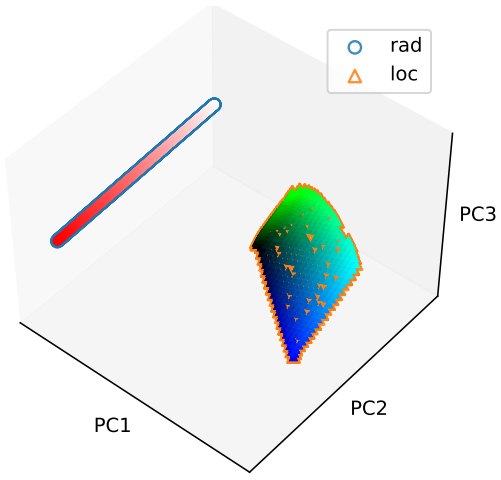


(f) Per-predicate PCAs (Plato + P^{2}BRS), including explained variance ratios.

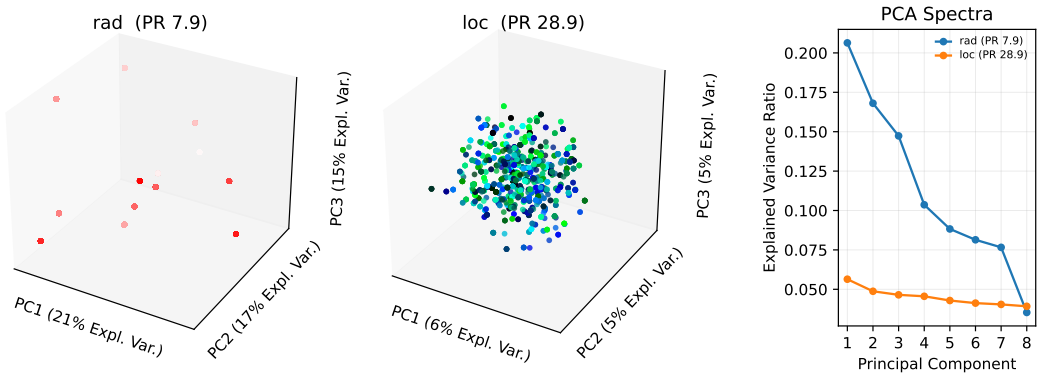
Figure 7: Principal component analyses of learned proposition embeddings in *FrankaCubesEnv*. Normalized parameters for a predicate instance $\mathsf{pos}(x, y, z)$, $\mathsf{rot}(\phi, \theta, \zeta)$, or $\mathsf{has}(r, g, b)$ are mapped to the RGB value $(r, g, b) \in [0, 1]^3$.



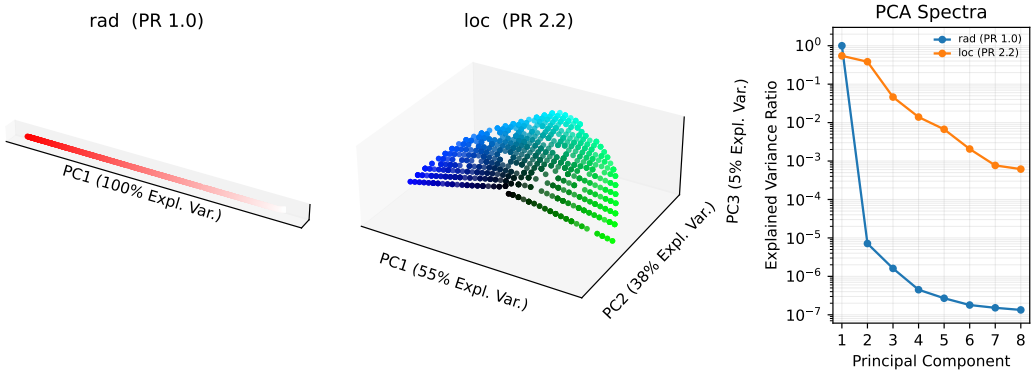
(a) Discretize



(b) Plato

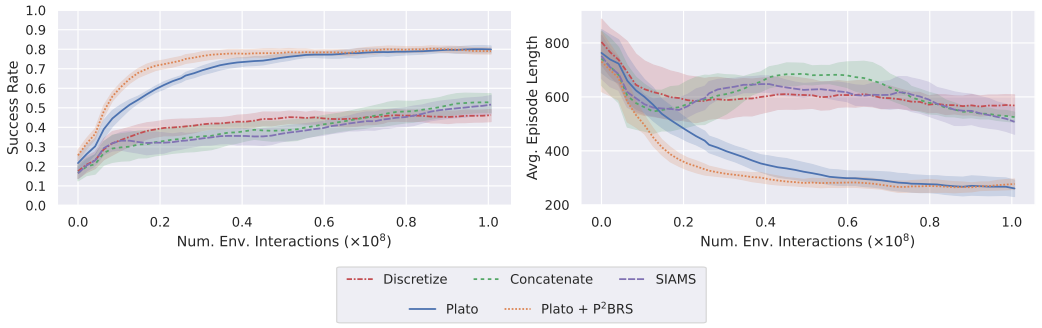


(c) Per-predicate PCAs (Discretize), including explained variance ratios.

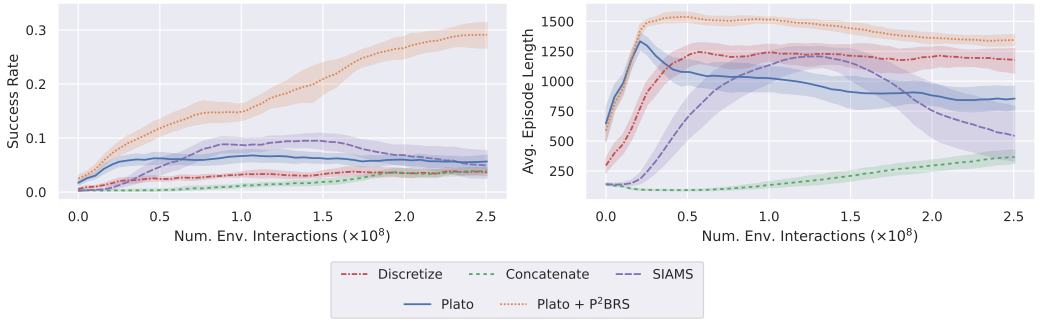


(d) Per-predicate PCAs (Plato), including explained variance ratios.

Figure 8: Principal component analyses of learned proposition embeddings in *FalloutWorld*. The normalized parameter for a predicate instance $\mathsf{rad}(\eta)$ is mapped to the RGB value $(1, 1-\eta, 1-\eta) \in [0,1]^3$ such that color transitions from white to red as radiation intensity threshold increases, and normalized parameters for an instance $\mathsf{loc}(x, y)$ are mapped to the RGB value $(0, x, y) \in [0,1]^3$. Note that $\mathsf{rad}(\eta)$ only yields 13 unique data points for Discretize, since it only has 13 bins under our discretization.

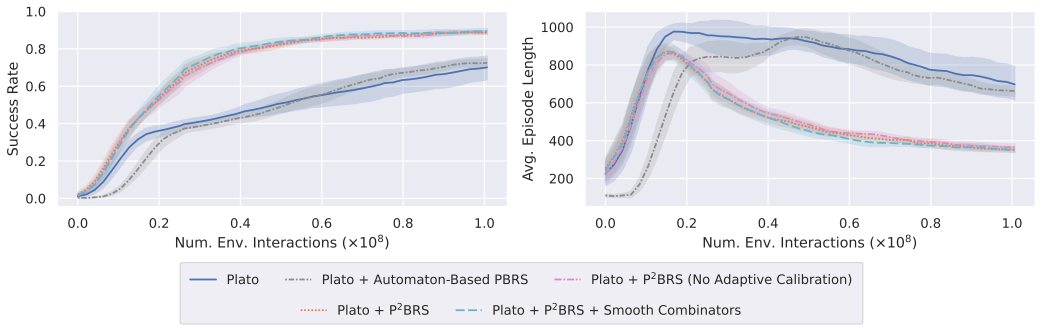


(a) *Go1RGBZoneEnv*

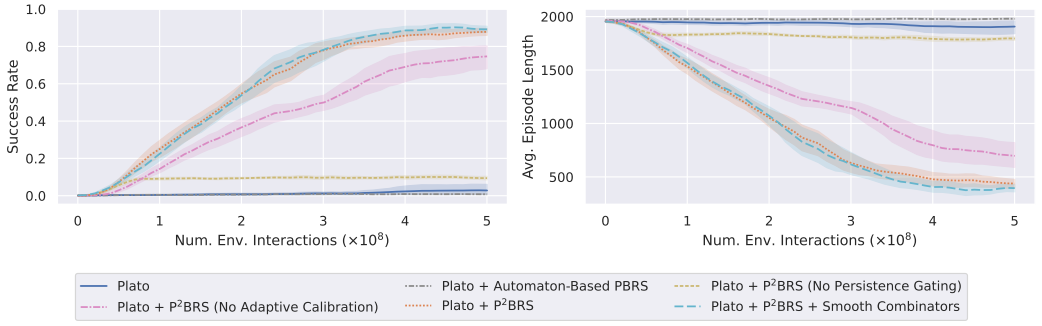


(b) *X2DronesEnv*

Figure 9: Evaluation curves for *finite*-horizon specifications using LTL2Action.



(a) *X2DronesEnv*



(b) *FrankaCubesEnv*

Figure 10: Evaluation curves for *finite*-horizon specifications under each P^{2}BRS ablation.