Mission-driven Exploration for Accelerated Deep Reinforcement Learning with Temporal Logic Task Specifications

Jun Wang JUNW@WUSTL.EDU
Washington University in St. Louis, 1 Brookings Drive St. Louis, MO 63130

Hosein Hasanbeig HHASANBEIG@MICROSOFT.COM
Microsoft Research, 300 Lafayette St, New York, NY 10012

Kaiyuan Tan KAIYUAN.TAN@VANDERBILT.EDU
Vanderbilt University, 2201 West End Ave, Nashville, TN 37235

Zihe Sun S.ZIHE@WUSTL.EDU
Washington University in St. Louis, 1 Brookings Drive St. Louis, MO 63130

Yiannis Kantaros IOANNISK@WUSTL.EDU
Washington University in St. Louis, 1 Brookings Drive St. Louis, MO 63130

Editors: N. Ozay, L. Balzano, D. Panagou, A. Abate

Abstract

This paper addresses the problem of designing control policies for agents with unknown stochastic dynamics and control objectives specified using Linear Temporal Logic (LTL). Recent Deep Reinforcement Learning (DRL) algorithms have aimed to compute policies that maximize the satisfaction probability of LTL formulas, but they often suffer from slow learning performance. To address this, we introduce a novel Deep Q-learning algorithm that significantly improves learning speed. The enhanced sample efficiency stems from a mission-driven exploration strategy that prioritizes exploration towards directions likely to contribute to mission success. Identifying these directions relies on an automaton representation of the LTL task as well as a learned neural network that partially models the agent-environment interaction. We provide comparative experiments demonstrating the efficiency of our algorithm on robot navigation tasks in unseen environments.[1]

Keywords: Reinforcement Learning, Temporal Logic Control Synthesis

1. Introduction

Deep Reinforcement Learning (DRL) has effectively synthesized control policies for autonomous systems under motion, sensing, and environmental uncertainty (Chen et al., 2017; Kiran et al., 2021; Zhou et al., 2023; Sachdeva et al., 2024). Typically, DRL specifies control objectives through reward functions. However, designing these rewards can be highly non-intuitive for complex tasks, and poorly constructed rewards may degrade performance (Dewey, 2014; Zhou and Li, 2022). To address this, Linear Temporal Logic (LTL) has been used to naturally encode complex tasks that would have been very hard to define using Markovian rewards e.g., consider a navigation task that requires visiting regions of interest in a specific order.

Several model-free DRL methods to design control policies satisfying LTL objectives have been proposed recently (Xu and Topcu, 2019; Gao et al., 2019; Hasanbeig et al., 2019; Lavaei et al., 2020; Kalagarla et al., 2021; Jiang et al., 2021; Bozkurt et al., 2020; Cai et al., 2021a; Wang et al., 2020;

[1] Hardware demonstrations and code are available on our project webpage: spec-driven-rl.github.io.

Jothimurugan et al., 2021; Bansal; Hasanbeig et al., 2022; Shao and Kwiatkowska, 2023). These approaches explore a product state space that expands exponentially with the state space size and task complexity, resulting in slow learning, further exacerbated by the sparse rewards used to design policies with probabilistic satisfaction guarantees Bozkurt et al. (2020). Model-based RL methods for LTL objectives have also been proposed in (Fu and Topcu, 2014; Brázdil et al., 2014; Cohen and Belta, 2021). These works use a learned MDP model to synthesize optimal policies within a finite number of iterations, achieving higher sample efficiency than model-free methods. However, these approaches are limited to MDPs with discrete state/action spaces, making them less suitable for applications that require handling continuous spaces.

In this paper, we present a new sample-efficient DRL algorithm to learn control policies for agents with LTL-encoded tasks. The agent-environment interaction is modeled as an unknown MDP with a continuous state space and discrete action space. Our proposed method builds on Deep Q-Networks (DQN) with LTL specifications (Mnih et al., 2013; Hasanbeig et al., 2020; Gao et al., 2019; Cai et al., 2021b) that typically employ ϵ-greedy policies. The key difference lies in our exploration strategy. Specifically, we propose a novel stochastic policy that extends ϵ-greedy policies, combining (i) an exploitation phase and (ii) a *mission-driven exploration* strategy. Instead of random exploration, our method prioritizes directions that may contribute to task satisfaction, leveraging the logical task structure. We demonstrate our algorithm's superior sample efficiency over DQN methods with ϵ-greedy policies and actor-critic methods on robot navigation tasks. We emphasize that the proposed stochastic policy can be coupled with any existing deep temporal difference learning method for LTL-encoded tasks (Gao et al., 2019; Cai et al., 2021b) as well as with any reward augmentation method that e.g., aims to assign non-zero rewards to intermediate goals (Icarte et al., 2022; Cai et al., 2022; Balakrishnan et al., 2022; Pathak et al., 2017; Hasanbeig et al., 2021; Zhang et al., 2023; Cai et al., 2023; Zhai et al., 2022) to further enhance sample-efficiency. The latter holds since our exploration strategy is agnostic to the reward structure.

A preliminary version of this work was presented in (Kantaros, October 2022; Kantaros and Wang, 2024), which introduced a similar exploration strategy to accelerate Q-learning for unknown MDPs with discrete state/action spaces. The core idea in (Kantaros, October 2022; Kantaros and Wang, 2024) was to use graph search techniques on a continuously learned MDP for guided exploration during training. In contrast, our current work addresses continuous state spaces, that are common in control systems, where learning and storing a full MDP model is computationally infeasible, making the graph search strategies of (Kantaros, October 2022; Kantaros and Wang, 2024) impractical. Also, unlike the proposed method, the previous works do not allow policy generalization to new environments due to the tabular representations of the action-value functions.

Contribution: *First*, we propose a new DQN algorithm for learning control policies for agents modeled as unknown MDPs with continuous state spaces and LTL-encoded tasks. *Second*, we show that the proposed control policy can be integrated with existing deep temporal difference learning methods for LTL tasks to enhance their sample efficiency. *Third*, we present comparative experiments that empirically demonstrate the sample efficiency of our method.

2. Problem Formulation

We consider an agent that is responsible for accomplishing a high-level task, expressed as an LTL formula, in an environment $\mathcal{W} \subseteq \mathbb{R}^d$, $d \in \{2, 3\}$ (Leahy et al., 2016; Guo and Zavlanos, 2017; Kantaros and Zavlanos, 2016). LTL is a formal language that comprises a set of atomic propositions (i.e., Boolean variables), denoted by $\mathcal{AP}$, Boolean operators, (i.e., conjunction $\wedge$, and negation $\neg$), and two temporal operators, next $\bigcirc$ and until $\mathcal{U}$. LTL formulas over a set $\mathcal{AP}$ can be constructed based on the following grammar: $\phi ::= \text{true} \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \,\mathcal{U}\, \phi_2$, where $\pi \in \mathcal{AP}$. For brevity, we abstain from presenting the derivations of other Boolean and temporal operators, e.g., *always* $\square$, *eventually* $\Diamond$, *implication* $\Rightarrow$, which can be found in (Baier and Katoen, 2008). We model the interaction of the agent with the environment $\mathcal{W}$ as an MDP:

Definition 1 (MDP) *An MDP is a tuple $\mathfrak{M} = (\mathcal{X}, \mathcal{A}, P, \mathcal{AP})$, where $\mathcal{X}$ is a continuous set of states; $\mathcal{A}$ is a finite set of actions. With slight abuse of notation $\mathcal{A}(x)$ denotes the available actions at state $x \in \mathcal{X}$; $P(x'|a, x)$ is a probability density function for the next state $x' \in \mathcal{X}$ given that the current MDP state and action is $x \in \mathcal{X}$ and $a \in \mathcal{A}$, respectively, capturing motion uncertainties. Also, $\int_{x'\in\mathcal{X}} P(x'|a, x) = 1$, for all $a \in \mathcal{A}(x)$; $\mathcal{AP}$ is a set of atomic propositions; $L : \mathcal{X} \to 2^{\mathcal{AP}}$ is the labeling function that returns the atomic propositions satisfied at a state $x \in \mathcal{X}$.*

In what follows, we consider atomic propositions of the form π^{r_i} that are true if the agent state is within a subspace $r_i \subseteq \mathcal{X}$ and false otherwise. For instance, consider the LTL formula $\phi = \Diamond(\pi^{r_1}) \wedge \Diamond(\pi^{r_2}) \wedge (\neg\pi^{r_1}\mathcal{U}\pi^{r_2}) \wedge \square(\neg\pi^{r_3})$ that requires an agent to eventually reach r_2 and r_1 in this order while always avoiding the unsafe sub-space r_3. We assume that the LTL formula is known to the agent before deployment.

Assumption 2 (MDP) *We assume the MDP is fully observable: at any time t the current state x_t and the observations $\ell_t = L(x_t) \in 2^{\mathcal{AP}}$, are known, while transition probabilities remain unknown.*

At any time step $T \geq 0$ we define (i) the agent's past path as $X_T = x_0x_1 \dots x_T$; (ii) the past sequence of observations as $L_T = \ell_0\ell_1 \dots \ell_T$, where $\ell_t \in 2^{\mathcal{AP}}$; (iii) the past sequence of features $\Psi_T = \psi(x_0)\psi(x_1)\dots\psi(x_T)$ and (iv) the past sequence of control actions $\mathcal{A}_T = a_0a_1 \dots a_{T-1}$, where $a_t \in \mathcal{A}(x_t)$. In (iii), the features $\psi_t = \psi(x_t)$, where $\psi : \mathcal{X} \to \Psi \subseteq \mathbb{R}^m$ may refer to MDP state x_t or high level semantic information (e.g., sensor measurements or distance to closest obstacle); their exact definition is application-specific (Faust et al., 2018). These four sequences can be composed into a complete past run, defined as $R_T = x_0\ell_0\psi_0a_0x_1\ell_1\psi_1a_1 \dots x_T\ell_T$. Let $\boldsymbol{\xi}$ be a finite-memory policy for $\mathfrak{M}$ defined as $\boldsymbol{\xi} = \xi_0\xi_1 \dots$, where $\xi_t : R_t \times \mathcal{A} \to [0, 1]$, and R_t is the past run for all $t \geq 0$. Let Ξ be the set of all such policies. Our goal is to develop a *sample-efficient algorithm* to learn a policy $\boldsymbol{\xi}^* \in \Xi$ that aims to maximize the probability of satisfying ϕ, i.e., $\mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}(\phi) = \mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}(\mathcal{R}_\infty : \mathcal{L}_\infty \models \phi)$, where $\mathcal{L}_\infty$ and $\mathcal{R}_\infty$ are sets collecting all possible sequences L_T and R_T of infinite horizon T (Baier and Katoen, 2008; Guo and Zavlanos, 2018). The problem is summarized as follows:

Problem 1 *Given a known LTL-encoded task specification ϕ, develop a sample-efficient DRL algorithm that can synthesize a finite memory control policy $\boldsymbol{\xi}^*$ for the unknown MDP that aims to maximize the satisfaction probability of ϕ.*

3. Accelerated Deep Reinforcement Learning with Temporal Logic Specifications

Building upon our earlier work (Kantaros, October 2022), we propose a new deep Q-learning algorithm that can quickly synthesize control policies for LTL-encoded tasks; see Alg. 1.

3.1. Converting LTL formulas into Automata

We first translate ϕ into the Deterministic Rabin Automaton (DRA) below [line 2, Alg.1].

Definition 3 (DRA (Baier and Katoen, 2008)) *A DRA over $2^{\mathcal{AP}}$ is a tuple $\mathfrak{D} = (\mathcal{Q}_D, q_D^0, \Sigma, \delta_D, \mathcal{F})$, where $\mathcal{Q}_D$ is a finite set of states; $q_D^0 \subseteq Q_D$ is the initial state; $\Sigma = 2^{\mathcal{AP}}$ is the input alphabet; $\delta_D : \mathcal{Q}_D \times \Sigma \to \mathcal{Q}_D$ is the transition function; and $\mathcal{F} : \{(\mathcal{G}_1, \mathcal{B}_1) \dots (\mathcal{G}_f, \mathcal{B}_f)\}$ is a set of accepting pairs where $\mathcal{G}_i, \mathcal{B}_i \subseteq \mathcal{Q}_D$, $\forall i \in \{1, \dots, f\}$.*

Algorithm 1 Accelerated Deep Q-Learning for LTL Tasks

```
**Input**: LTL formula $\phi$
Initialize: $Q^{\boldsymbol{\mu}}(\psi(s), a; \theta)$ arbitrarily; Replay memory $\mathcal{M}$
Translate $\phi$ into a DRA $\mathfrak{D}$
Construct distance function $d_\phi$ over DRA
Train a NN $g : \Psi \times \mathcal{X} \rightarrow \mathcal{A}$
$\boldsymbol{\mu} = (\epsilon, \delta)$-greedy($Q^{\boldsymbol{\mu}}$)
**for** $\texttt{episode} = 1$ to $M$ **do**
    Sample environment $\mathcal{W}_k$ and initial state $x_0 \in \mathcal{X}$, and construct $\psi_{\mathfrak{P}}(s_0) = (\psi(x_0), q_D^0)$
    **for** $t = 0$ to $T_{\text{max}}$ **do**
        Pick and execute action $a_t$
        Observe $s_{t+1} = (x_{\text{next}}, q_D^{t+1})$, $\psi_{\mathfrak{P}}(s_{t+1})$, and $r_t$
        Store transition $(\psi_{\mathfrak{P}}(s_t), a_t, r_t, \psi_{\mathfrak{P}}(s_{t+1}))$ in $\mathcal{M}$
        Sample a batch of $(\psi_{\mathfrak{P}}(s_n), a_n, r_n, \psi_{\mathfrak{P}}(s_{n+1}))$ from $\mathcal{M}$
        Set $y_t = r_t + \gamma \max_{a'} Q^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s_{t+1}), a'; \theta)$
        $\theta_{t+1} = \theta_t + \alpha(r_t + \gamma \max_{a'} Q(\psi_{\mathfrak{P}}(s'), a'; \theta_t) - Q(\psi_{\mathfrak{P}}(s), a; \theta))\nabla_\theta Q(\psi_{\mathfrak{P}}(s), a; \theta)$
        Update the policy $\boldsymbol{\mu}$
        $s_t \leftarrow s_{t+1}$
    **end for**
**end for**
```

We note that any LTL formula can be translated into a DRA. To define the accepting condition of a DRA, we need to introduce the following concepts. First, an infinite run $\rho_D = q_D^0 q_D^1 \dots q_D^t \dots$ over an infinite word $\sigma = \sigma_0\sigma_1\sigma_2 \dots \in \Sigma^\omega$, where $\sigma_t \in \Sigma$, $\forall t \in \mathbb{N}$, is an infinite sequence of DRA states q_D^t, $\forall t \in \mathbb{N}$, such that $\delta_D(q_D^t, \sigma_t) = q_D^{t+1}$. An infinite run ρ_D is called *accepting* if there exists at least one pair $(\mathcal{G}_i, \mathcal{B}_i)$ such that $\text{Inf}(\rho_D) \bigcap \mathcal{G}_i \neq \emptyset$ and $\text{Inf}(\rho_D) \bigcap \mathcal{B}_i = \emptyset$, where $\text{Inf}(\rho_D)$ represents the set of states that appear in ρ_D infinitely often.

3.2. Product MDP

Given MDP $\mathfrak{M}$ and non-pruned DRA $\mathfrak{D}$, we define product MDP (PMDP) $\mathfrak{P} = \mathfrak{M} \times \mathfrak{D}$ as follows.

Definition 4 (PMDP) *Given an MDP $\mathfrak{M} = (\mathcal{X}, \mathcal{A}, P, \mathcal{AP})$ and a DRA $\mathfrak{D} = (\mathcal{Q}_D, q_D^0, \Sigma, \delta_D, \mathcal{F})$, we define the PMDP $\mathfrak{P} = \mathfrak{M} \times \mathfrak{D}$ as $\mathfrak{P} = (\mathcal{S}, \mathcal{A}_{\mathfrak{P}}, \mathcal{P}_{\mathfrak{P}}, \mathcal{F}_{\mathfrak{P}})$, where (i) $\mathcal{S} = \mathcal{X} \times \mathcal{Q}_D$ is the set of states, $s = (x, q_D) \in \mathcal{S}$, $x \in \mathcal{X}$, and $q_D \in \mathcal{Q}_D$; (ii) $\mathcal{A}_{\mathfrak{P}}$ is the set of actions inherited from the MDP, $\mathcal{A}_{\mathfrak{P}}(s) = \mathcal{A}(s)$, where $s = (x, q_D)$; (iii) $\mathcal{P}_{\mathfrak{P}}$ is the probability density function, $\mathcal{P}_{\mathfrak{P}}(s'|s, a_P) = \mathcal{P}(x'|x, a)$, where $s = (x, q_D) \in \mathcal{S}$, $s' \in (x', q_D') \in \mathcal{S}$, $a_P \in \mathcal{A}(s)$ and $q_D' = \delta_D(q, L(x))$; (iv) $\mathcal{F}_{\mathfrak{P}} = \{\mathcal{F}_i^{\mathfrak{P}}\}_{i=1}^f$ is the set of accepting states, where $\mathcal{F}_i^{\mathfrak{P}} = \mathcal{X} \times \mathcal{F}_i$ and $\mathcal{F}_i = (\mathcal{G}_i, \mathcal{B}_i)$.*

Given any policy $\boldsymbol{\mu} : \mathcal{S} \rightarrow \mathcal{A}_{\mathfrak{P}}$ for $\mathfrak{P}$, we define an infinite run $\rho_{\mathfrak{P}}^{\boldsymbol{\mu}}$ of $\mathfrak{P}$ to be an infinite sequence of states of $\mathfrak{P}$, i.e., $\rho_{\mathfrak{P}}^{\boldsymbol{\mu}} = s_0 s_1 s_2 \dots$, where $P_{\mathfrak{P}}(s_{t+1}|s_t, \boldsymbol{\mu}(s_t)) > 0$. An infinite run $\rho_{\mathfrak{P}}^{\boldsymbol{\mu}}$ is accepting, i.e., $\boldsymbol{\mu}$ satisfies ϕ with a non-zero probability if $\text{Inf}(\rho_{\mathfrak{P}}^{\boldsymbol{\mu}}) \bigcap \cap \mathcal{G}_i^{\mathfrak{P}} \neq \emptyset$ and $\text{Inf}(\rho_{\mathfrak{P}}^{\boldsymbol{\mu}}) \bigcap \mathcal{B}_i^{\mathfrak{P}} = \emptyset$, $\forall i \in \{1, \dots, f\}$, where $\mathcal{G}_i^{\mathfrak{P}} = \mathcal{G}_i \times \mathcal{X}$ and $\mathcal{B}_i^{\mathfrak{P}} = \mathcal{B}_i \times \mathcal{X}$.

3.3. Accelerated Policy Learning for LTL-encoded Tasks

In this section, we present our accelerated DRL algorithm to solve Problem 1. The proposed algorithm is summarized in Alg. 1. The output of Alg. 1 is a stationary and deterministic policy $\boldsymbol{\mu}^*$ for the PMDP $\mathfrak{P}$. Projection of $\boldsymbol{\mu}^*$ onto the MDP $\mathfrak{M}$ yields the finite memory policy $\boldsymbol{\xi}^*$.

We apply episodic DRL to compute $\boldsymbol{\mu}^*$. Since the test-time deployment environment may be unavailable during training, we train the policy across multiple environments to ensure generalization. Specifically, we use $K > 0$ environments $\mathcal{W}_k$, $k \in \{1, \dots, K\}$; e.g., in robot navigation tasks, these environments may differ in their geometric structure. To help the policy distinguish and adapt to different environments, we leverage features $\psi : \mathcal{X} \rightarrow \Psi$ related to the agent state (e.g., distances to obstacles or goal regions). Thus, the policy $\boldsymbol{\mu}^*$ is designed so that it maps features, denoted by $\psi_{\mathfrak{P}}(s_t)$ (instead of states s_t) to actions a, where $s_t = (x_t, q_D^t)$ and $\psi_{\mathfrak{P}}(s_t) = (\psi(x_t), q_D^t)$.

First, we design a reward function to motivate the agent to satisfy the PMDP's accepting condition. Specifically, we adopt the reward function $R : \Psi \times \mathcal{A}_{\mathfrak{P}} \times \Psi \rightarrow \mathbb{R}$ which given a transition $(\psi(s), a_{\mathfrak{P}}, \psi(s'))$ returns a reward as follows (Gao et al., 2019): $r_{\mathcal{G}}$, if $\psi(s') \in \mathcal{G}_i^{\mathfrak{P}}$; $r_{\mathcal{B}}$, if $\psi(s') \in \mathcal{B}_i^{\mathfrak{P}}$; r_d, if $\psi(s')$ is a deadlock state (i.e., a state with no outgoing transitions); and r_o otherwise, where $\forall i \in \{1, \dots, f\}, r_{\mathcal{G}} > r_{\mathcal{B}} > 0, r_d < r_0 \leq 0$.[2] The policy $\boldsymbol{\mu}^*$ is designed so that it maximizes the expected accumulated return, i.e., $\boldsymbol{\mu}^*(\psi_{\mathfrak{P}}(s)) = \text{argmax}_{\boldsymbol{\mu} \in \mathcal{D}}\, U^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s))$, where $\mathcal{D}$ is the set of all stationary deterministic policies over $\mathcal{S}$, and $U^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s)) = \mathbb{E}^{\boldsymbol{\mu}}[\sum_{t=0}^{\infty} \gamma^t R(\psi_{\mathfrak{P}}(s_t), a_t, \psi_{\mathfrak{P}}(s_{t+1}))$ $|\psi_{\mathfrak{P}}(s) = \psi_{\mathfrak{P}}(s_0)]$. In the latter equation, $\mathbb{E}^{\boldsymbol{\mu}}[\cdot]$ denotes the expected value given that PMDP actions are selected as per the policy $\boldsymbol{\mu}$, $0 \leq \gamma < 1$ is the discount factor, and $s_0, \dots, s_t$ is the sequence of states generated by policy $\boldsymbol{\mu}$ up to time step t, initialized at s_0. The training process terminates when the action value function $Q^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s), a)$ has converged. This value function is defined as the expected return for taking action a when $\psi_{\mathfrak{P}}(s)$ is observed and then following the policy $\boldsymbol{\mu}$, i.e., $Q^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s), a) = \mathbb{E}^{\boldsymbol{\mu}}[\sum_{t=0}^{\infty} \gamma^t R(\psi_{\mathfrak{P}}(s_t), a_t, \psi_{\mathfrak{P}}(s_{t+1}))|s_0 = s, a_0 = a]$. To handle the continuous state space, we employ a NN to approximate the action-value function $Q^{\boldsymbol{\mu}}(\psi_{\mathfrak{P}}(s), a)$ denoted by $Q(\psi_{\mathfrak{P}}(s), a; \theta)$ where θ denotes the Q-network weights.

At the beginning of each episode (i.e., $t = 0$) we sample an environment $\mathcal{W}_k$ and initial system state $x_0 \in \mathcal{X}$. The latter combined with an initial DRA state q_D^0 yields an initial PMDP state $s_0 = [x_0, q_D^0]$ along with $\psi_{\mathfrak{P}}(s_0)$ [lines 7-8, Alg. 1]. Then, at each time step t of the episode, we apply an action a_t selected as per a stochastic policy $\boldsymbol{\mu}$; its definition will be provided later in the text [lines 6 & 10, Alg. 1]. Then, we observe the next state s_{t+1} and reward $r_t = R(\psi_{\mathfrak{P}}(s_t), a_t, \psi_{\mathfrak{P}}(s_{t+1}))$ [line 11, Alg. 1]. This transition is stored in a memory log $\mathcal{M}$ [line 12, Alg. 1]. Then, a random batch of transitions of the form $(\psi_{\mathfrak{P}}(s_n), a_n, r_n, \psi_{\mathfrak{P}}(s_{n+1}))$ is sampled from $\mathcal{M}$ [line 13, Alg. 1] that is used to update the Q-network weights θ [line 15, Alg. 1]. Particularly, the Q-network is trained by minimizing the loss function $L(\theta) = \mathbb{E}_{s,a \sim \boldsymbol{\mu}}[(y - Q(\psi_{\mathfrak{P}}(s), a; \theta))^2]$ where $y = \mathbb{E}_{s' \sim \boldsymbol{\mu}}[r + \gamma \max_{a'} Q(\psi_{\mathfrak{P}}(s'), a'; \theta)|s, a]$. The weights θ are updated by applying gradient descent on $L(\theta)$, i.e., $\theta_{t+1} = \theta_t + \alpha(r_t + \gamma \max_{a'} Q(\psi_{\mathfrak{P}}(s'), a'; \theta_t) - Q(\psi_{\mathfrak{P}}(s), a; \theta)) \nabla_{\theta} Q(\psi_{\mathfrak{P}}(s), a; \theta)$. Next, we update the policy $\boldsymbol{\mu}$ [line 16, Alg. 1] and the episode run continues [line 17, Alg. 1]. An episode terminates either when a maximum number $T_{\text{max}} > 0$ of steps has been made or when a terminal/deadlock state is visited (i.e., a PMDP state with no outgoing transitions); reaching these deadlock states implies violation of the LTL formula.

As a policy $\boldsymbol{\mu}$, we use the (ϵ, δ)-greedy policy in (Kantaros, October 2022). In this policy, (i) the *greedy* action $a^* = \arg\max_{a \in \mathcal{A}_{\mathfrak{P}}} Q(s, a; \theta)$ is chosen with probability $1 - \epsilon$, as in standard ϵ-greedy approach. (ii) With probability $\epsilon = \delta_b + \delta_e$, an exploratory action is taken, defined as follows: (ii.1) a *random* action is selected with probability δ_e; and (ii.2) a *biased* action a_b, that will most likely drive the agent towards an accepting product state in $\mathcal{G}_i^{\mathfrak{P}}$, is chosen with probability δ_b (defined in Section 3.4). The parameter ϵ ensures all actions are explored infinitely often, eventually decaying to 0. The policy $\boldsymbol{\mu}$ is updated by recalculating a^* and a_b and decreasing ϵ [line 16, Alg. 1],

[2] Defining rewards so that maximization of the expected accumulated reward is equivalent to maximization of the satisfaction of probability is out of the scope of this work. However, our proposed exploration strategy is reward agnostic and can be coupled with any reward functions and discount factors that have been proposed to model satisfaction probabilities of LTL formulas; c.f., (Hasanbeig et al., 2019; Bozkurt et al., 2020).

converging to a deterministic policy greedy for the state-action value function. The ϵ-greedy policy is a special case of this approach when $\delta_b = 0$. Section 4 details our choice of δ_b and δ_e.

3.4. Computation of the Biased Action

Next, we describe how the biased action a_b is computed. We first prune the DRA by removing infeasible DRA transitions as in (Kantaros and Wang, 2024). Then we introduce a distance-like function over the DRA state-space that computes how far any given DRA state is from the sets of accepting states $\mathcal{G}_i$ (Kantaros and Zavlanos, 2020; Kantaros et al., 2022). Intuitively, this function measures how 'far' the agent is from accomplishing an LTL task. Let SP_{q_D,q'_D} be the shortest path (in terms of the number of hops) in the pruned DRA from q_D to q'_D and $|SP_{q_D,q'_D}|$ denote its cost (number of hops). Then, we define the function d as: $d(q_D, q'_D) = |SP_{q_D,q'_D}|$, if SP_{q_D,q'_D}exists ; otherwise, $d(q_D, q'_D) = \infty$. We define a function measuring the distance of any DRA state q_D to the set of accepting pairs: $d_\phi(q_D, \mathcal{F}) = \min_{q_D^G \in \bigcup_{i\in\{1,\dots,f\}} \mathcal{G}_i} d(q_D, q_D^G)$.

Let $s_t = (x_t, q_D^t)$ denote the current PMDP state in Alg 1. Given s_t, let $\mathcal{Q}_{\text{goal}}(q_D^t) \subset \mathcal{Q}_D$ be a set collecting all DRA states that are one-hop reachable from q_D^t in the pruned DRA and closer to the accepting DRA states than q_D^t is as per d_ϕ. Formally, we define $\mathcal{Q}_{\text{goal}}(q_D^t) = \{q'_D \in \mathcal{Q}_D | (\exists \sigma \in \Sigma_{\text{feas}} \text{ s.t. } \delta_D(q_D^t, \sigma) = q'_D) \wedge (d_\phi(q'_D, \mathcal{F}) = d_\phi(q_D^t, \mathcal{F}) - 1)\}$, where $\Sigma_{\text{feas}} \subseteq \Sigma$ collects all feasible symbols, i.e., symbols that can be generated without requiring the agent to be in multiple states simultaneously (Kantaros and Zavlanos, 2020). Among all states in $\mathcal{Q}_{\text{goal}}(q_D^t)$, we select one randomly, denoted by q_{goal}. Next, given q_{goal}, we define the set $\mathcal{X}_{\text{goal}}(q_D^t) = \{x \in \mathcal{X} | \delta_D(q_D^t, L(x)) = q_{\text{goal}} \in \mathcal{Q}_{\text{goal}}(q_D^t)\}$ collecting all MDP states that if the agent eventually reaches, transition from s_t to $s_{\text{goal}} = [x_{\text{goal}}, q_{\text{goal}}]$ will occur. Among all states in $\mathcal{X}_{\text{goal}}(q_D^t)$, we pick one randomly as x_{goal}.

Given the current state x_t and the goal state x_{goal}, our goal is to select the action a_b to be the one that will most likely drive the agent 'closer' to x_{goal} than it currently is. The key challenge in computing a_b is that it requires knowledge of the MDP transition probabilities which are unknown. Learning the MDP transition probabilities is memory inefficient and computationally expensive even for discrete state spaces, as in (Kantaros, October 2022), let alone for continuous spaces considered here. An additional challenge, compared to (Kantaros, October 2022), is to design an appropriate proximity metric between the states x_{goal} and x_t.

To address these challenges, we design a neural network (NN) model $g : \Psi \times \mathcal{X} \rightarrow \mathcal{A}$ that (partially) captures the agent dynamics and it is capable of outputting the action a_b, given as input features $\psi(x_t) \in \Psi$ observed/generated at an MDP state x_{start} (i.e., x_t during the RL training phase) and a goal MDP state x_{goal}. Notice that g takes as input the features $\psi(x_{\text{start}})$, instead of x_{start}, to ensure its generalization (up to a degree) to unseen environments. This NN is trained before the RL training phase [line 5, Alg. 1]. Given a training dataset with data points of the form $(\psi(x_{\text{start}}), x_{\text{goal}}, a_b)$ we train the NN g so that a cross-entropy loss function, denoted by $L(a_{\text{gt}}, a_{\text{pred}})$, is minimized, where a_{gt} is the ground truth label and a_{pred} is NN-generated prediction.

Next, we discuss how we generate the training dataset; see Alg. 2. First, we discretize the state space $\mathcal{X}$ in m disjoint cells $\mathcal{C}_i \subseteq \mathcal{X}$, i.e., (a) $\text{int}(\mathcal{C}_i) \cap \text{int}(\mathcal{C}_j) = \emptyset, \forall i \neq j$ (where $\text{int}(\mathcal{C}_i)$ denotes the interior of $\mathcal{C}_i$); and (b) $\cup_{i=1}^m \mathcal{C}_i = \mathcal{X}$ [line 2, Alg. 2]. This space discretization is performed only to simplify the data collection process so that the (training) goal states x_{goal} are selected from a discrete set; the goal states x_{goal} are the centers of the cells $\mathcal{C}_i$. For simplicity, we also require that (c) each region $r_e \subseteq \mathcal{X}$ appearing in the formula ϕ in a predicate π^{r_e} to be fully inside a cell $\mathcal{C}_i$.

Then we select a training environment $\mathcal{W}_k$, $k \in \{1, \dots, K\}$ [line 3, Alg. 2]. The following steps are repeated for every training environment $\mathcal{W}_k$. We define a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, w\}$, where $\mathcal{V}$ is a set

Algorithm 2 Building a Training Dataset for NN g

```
1: Input: state space $\mathcal{X}$
2: Discretize $\mathcal{X}$ into $m > 0$ cells $\mathcal{C}_i \subset \mathcal{X}$
3: for $\mathcal{W}_k$, $k \in \{1, \dots, K\}$ do
4:     Construct the graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, w\}$ and the set $\mathcal{V}_{\text{avoid}}$;
5:     Sample $M$ states $x_{\text{start}} \in \mathcal{X}$ and collect them in $\mathcal{X}_{\text{start}} \subseteq \mathcal{X}$
6:     Initialize $n(x_{\text{start}}, a) = 0, \forall x_{\text{start}} \in \mathcal{X}_{\text{start}}, a \in \mathcal{A}$ ;
7:     for $x_{\text{start}} \in \mathcal{X}_{\text{start}}$ do
8:         for $i_{\text{goal}} \in \mathcal{V}$ do
9:             Define goal state $x_{\text{goal}}$ as the center of $\mathcal{C}_{i_{\text{goal}}}$;
10:            for $a \in \mathcal{A}$ do
11:                Initialize $D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) = 0$;
12:                for $z = 1$ to $Z$ do
13:                    Simulate next state $x_{\text{next}}$ if $x_t = x_{\text{start}}$, $a_t = a$;
14:                    Compute $i_{\text{next}}$;
15:                    if $i_{\text{next}} \notin \mathcal{V}_{\text{avoid}}$ then
16:                        $n(x_{\text{start}}, a) \leftarrow n(x_{\text{start}}, a) + 1$;
17:                        Compute $d_{\mathcal{W}}(x_{\text{start}}, a, x_{\text{goal}}) = d_{\mathcal{G}}(i_{\text{next}}, i_{\text{goal}})$;
18:                        $D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) \leftarrow D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) + d_{\mathcal{W}}(x_{\text{start}}, a, x_{\text{goal}})$
19:                    end if
20:                end for
21:            end for
22:        end for
23:        Compute $p(x_{\text{start}}, a) = n(x_{\text{start}}, a)/K, \forall a \in \mathcal{A}$
24:        Compute $\bar{D}_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) = D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})/K, \forall a$
25:        Select a $\zeta \in [0, p_{\text{max}}(x_{\text{start}}) - p_{\text{min}}(x_{\text{start}})]$,
26:        Construct $\mathcal{A}_{\text{safe}}(x_{\text{start}}) = \{a \in \mathcal{A} | p(x_{\text{start}}, a) \geq p(x_{\text{start}}) - \zeta\}$;
27:        Compute $a_b = \text{argmin}_{a \in \mathcal{A}_{\text{safe}}(x_{\text{start}})} \bar{D}_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})$
28:        Add the datapoint $(x_{\text{start}}, x_{\text{goal}}, a_b)$ to the training dataset.
29:    end for
30: end for
```

of nodes indexed by the cells $\mathcal{C}_i$ and $\mathcal{E}$ is a set of edges (each edge connects cells sharing a common boundary) [line 4, Alg. 2]. We collect in a set $\mathcal{V}_{\text{avoid}} \subseteq \mathcal{V}$ all graph nodes i that are associated with sub-spaces $r_i \subset \mathcal{X}$ that should always be avoided as per an LTL formula ϕ of interest. We exclude edges between adjacent nodes if at least one of them belongs to $\mathcal{V}_{\text{avoid}}$. We require the discretization to yield a connected graph $\mathcal{G}$. Next, we define the function $w : \mathcal{E} \rightarrow \mathbb{R}$ assigning a weight w_{ij} to the edge connecting nodes i and j. These weights are user-specified designed to capture the traveling cost from i to j; e.g., in Section 4.1, we use the ℓ_2 distance. Given w, we can compute the weighted shortest path over $\mathcal{G}$ connecting any two nodes i and j. We denote the shortest path cost by $d_{\mathcal{G}}(i, j)$.

Given $\mathcal{G}$, we perform the following steps to construct $M > 0$ datapoints of the form $(\psi(x_{\text{start}}), x_{\text{goal}}, a_b)$ associated with $\mathcal{W}_k$. **(i)** First, we sample M system states $x_{\text{start}} \in \mathcal{X}$ from the continuous state space [line 5, Alg. 2] and compute features $\psi_{\text{start}} = \psi(x_{\text{start}})$. Also, we initialize a counter $n(\psi_{\text{start}}, a) = 0$ for all ψ_{start} and actions a [line 6, Alg. 2]; $n(\psi_{\text{start}}, a)$ will be defined in a later step. **(ii)** Then, we repeat the following steps for all ψ_{start} and each goal state x_{goal} [line 7-9, Alg. 2]. (ii.1) We pick an action $a \in \mathcal{A}$ and we initialize $D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) = 0$ for all $a \in \mathcal{A}$ ($D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})$ captures the average distance from x_{start} to a fixed goal state x_{goal} once a is applied) [line 11, Alg. 2]. (ii.2) Given the action a, we simulate the next state (i.e., at time $t + 1$), denoted by x_{next}, assuming that at time t the system is in state x_{start} and applies action a [line 13, Alg. 2]. This step requires access to either a simulator of the system dynamics f or the actual system which is a common requirement in RL. (ii.3) We compute the node $i_{\text{next}} \in \mathcal{V}$ for which it holds $x_{\text{next}} \in \mathcal{C}_{i_{\text{next}}}$ [line 14, Alg. 2]. (ii.4) We increase the counter $n(\psi_{\text{start}}, a)$ by 1 if $i_{\text{next}} \notin \mathcal{V}_{\text{avoid}}$ [line 15-16, Alg. 2]. (ii.5) We compute the distance to the goal state x_{goal} once the action a is applied at state x_{start} [line 17, Alg. 2]. With slight abuse of notation, we denote this distance as $d_{\mathcal{W}}(x_{\text{start}}, a, x_{\text{goal}}) = d_{\mathcal{G}}(i_{\text{next}}, i_{\text{goal}})$, where i_{goal} is the index of the cell whose center is x_{goal}. Then, we compute $D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) \leftarrow D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) + d_{\mathcal{W}}(x_{\text{start}}, a, x_{\text{goal}})$ [line 18, Alg. 2]; We repeat the steps (ii.2)-(ii.5) for $Z > 0$ times [line 12, Alg. 2]. Once this is completed, we repeat (ii.1)-(ii.5) for each possible action a [line 10, Alg. 2]. **(iii)** Once (ii) is completed, we compute (iii.1) $p(\psi_{\text{start}}, a) = n(\psi_{\text{start}}, a)/Z$ for all ψ_{start} and actions $a \in \mathcal{A}$ [line 23, Alg. 2] and (iii.2) $\bar{D}_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}}) = D_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})/Z$ [line 24, Alg. 2], where $\bar{D}_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})$ estimates the probability that a safe state will be reached after applying the action a when the features ψ_{start} are observed (i.e., $i_{\text{next}} \notin \mathcal{V}_{\text{avoid}}$). **(iv)** Once (iii) is completed, (iv.1) we compute $p_{\text{min}}(\psi_{\text{start}}) = \min_{a \in \mathcal{A}} p(\psi_{\text{start}}, a)$ for each ψ_{start}; this corresponds to the probability $p(\psi_{\text{start}}, a)$ of the least 'safe' action a when ψ_{start} is observed. Similarly, we can define $p_{\text{max}}(\psi_{\text{start}}) = \max_{a \in \mathcal{A}} p(\psi_{\text{start}}, a)$. (iv.2) We compute the set $\mathcal{A}_{\text{safe}}(\psi_{\text{start}}) = \{a \in \mathcal{A} \mid p(\psi_{\text{start}}, a) \geq p_{\text{max}}(\psi_{\text{start}}) - \zeta\}$, for some $\zeta \in [0, p_{\text{max}}(\psi_{\text{start}}) - p_{\text{min}}(\psi_{\text{start}})]$ [lines 25-26, Alg. 2]. This set of actions is non-empty by definition of ζ and collects all actions that are considered 'safe' enough (as per ζ) at ψ_{start}. (iv.3) Given ψ_{start} and a goal state x_{goal}, we define the biased action $a_b = \text{argmin}_{a \in \mathcal{A}_{\text{safe}}(x_{\text{start}})} \bar{D}_{\mathcal{G}}(x_{\text{start}}, a, x_{\text{goal}})$ [line 27, Alg. 2]. This results in a training data-point $(\psi_{\text{start}}, x_{\text{goal}}, a_b)$ [line 28, Alg. 2]. Trained NN g is used as $a_b = g(\psi(x_t), x_{\text{goal}})$.

Remark 5 (Training Dataset to Bootstrap RL) *The training dataset collected to train the NN g can also be leveraged to bootstrap the RL training phase. For instance, the collected pairs of states and actions can be augmented with their corresponding rewards to initialize the replay memory $\mathcal{M}$.*

4. Experiments

We conduct experiments on robot navigation tasks. Our method outperforms related approaches in sample efficiency, especially as task or environmental complexity increases. All methods are tested on a GeForce RTX 3080 GPU with 64 GB RAM.

4.1. Setting Up Experiments

Simulator: We consider a robot with unknown differential drive dynamics (as in (Schlotfeldt et al., 2018)). The system state is $x_t = [p_t^1, p_t^2, \theta_t]^T \in \mathbb{R}^3$ modeling position and orientation. The control input consists of linear velocity $u_t \in [-0.26, 0.26]$ m/s and angular velocity $\omega_t \in [-1.82, 1.82]$ rad/s for all $t \geq 0$. We consider additive actuation noise, i.e., the control input becomes $\bar{u}_t = u_t + w_t^u$ and $\bar{\omega}_t = \omega_t + w_t^\omega$, with $w_t^u, w_t^\omega \sim \mathcal{N}(2e^{-3}, 1e^{-3})$ representing Gaussian noise. The action set $\mathcal{A}$ includes 23 combinations of linear and angular velocities. We define the feature function ψ as $\psi(x_t) = [\ell_t^1, \rho_t^1, \ell_t^2, \rho_t^2, p_t^1, p_t^2, \theta_t]^T \in \mathbb{R}^7$, where ℓ_t^1 and ρ_t^1 represent the distance and angle to the nearest obstacle at time t, and ℓ_t^2 and ρ_t^2 are defined similarly for the second closest obstacle.

Environments: We evaluate our method on two distinct groups of environments, each comprising 4 training and 4 testing environments with varying obstacle sizes and placements. Group A contains 3 to 5 cylindrical obstacles per environment, while Group B contains 10 to 12. The training and test environments for each group share the same goal locations.

BiasedNN: Because LTL tasks are defined over $\mathcal{W}$, we discretize it into a 12×12 grid (144 cells $\mathcal{C}_i \subset \mathcal{W}$), yielding a graph $\mathcal{G}$ with $|\mathcal{V}| = 144$ nodes. We fix $Z = 20$ and $\zeta = 0.1$, and assign each edge weight w_{ij} as the Euclidean distance between nodes i and j. The network g takes a 9-dimensional input vector$[\psi(x_t), x_{\text{goal}}] \in \mathbb{R}^9$ (where $\psi(x_t)$ is the 7-dimensional state embedding and x_{goal} are the 2-D goal coordinates). Its output is a 23-way softmax over the action set. The architecture comprises two ReLU-activated fully connected layers $[9, 2048, 1024, 23]$. The network is trained for 50 epochs using Adam at a learning rate of $1e^{-3}$.

Policy Parameters and Rewards: We initialize $\delta_b = \delta_e = 0.5$. Both δ_b and δ_e decay linearly with the number of episodes; δ_b decays 1.25 times faster than δ_e, and after $200,000$ episodes, it is set to 0. Also, we select $\gamma = 0.99$ and $r_{\mathcal{G}} = 100$, $r_{\mathcal{B}} = 10$, $r_o = -0.01$, and $r_d = -100$.

Baselines: We consider (i) DQN with ϵ-greedy (Gao et al., 2019), (ii) PPO (Schulman et al., 2017), and (iii) SAC (Haarnoja et al., 2018), each applied to the PMDP with the same reward function R. We use the same decay rate for ϵ, set $T_{\max} = 500$ steps, and start with an empty replay buffer for both our method and DQN. We train NN g for T_g hours and run Alg 1 with (ϵ, δ)-greedy policy for $T_{\epsilon,\delta}$ hours, respectively. The training time for all baselines is $T_g + T_{\epsilon,\delta}$ hours.

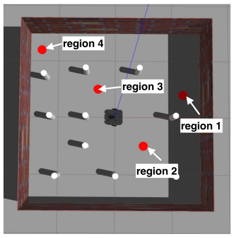


Figure 1: Example Environment

Evaluation Metrics: We evaluate sample efficiency using three metrics: *(i) Average return per episode*: $G = \sum_{t=0}^{T_{\max}} \gamma^t R(\psi(s_t), \boldsymbol{\mu}(\psi(s_t)), \psi(s_{t+1}))$, where higher G indicates better training performance. *(ii) Test-time accuracy of the trained RL controllers in the training environments* $\mathcal{W}_k$. *(iii) Test-time accuracy in unseen environments*. To compute (ii)-(iii), we randomly select 120 initial agent states. Then, for each initial state, we let the agent navigate the environment using the learned controller for 500 time steps. A run is considered 'successful' if the trajectory reaches the accepting DRA states at least twice without hitting any deadlock DRA states. We use the percentage of successful runs as the accuracy of the learned controller upon evaluation.

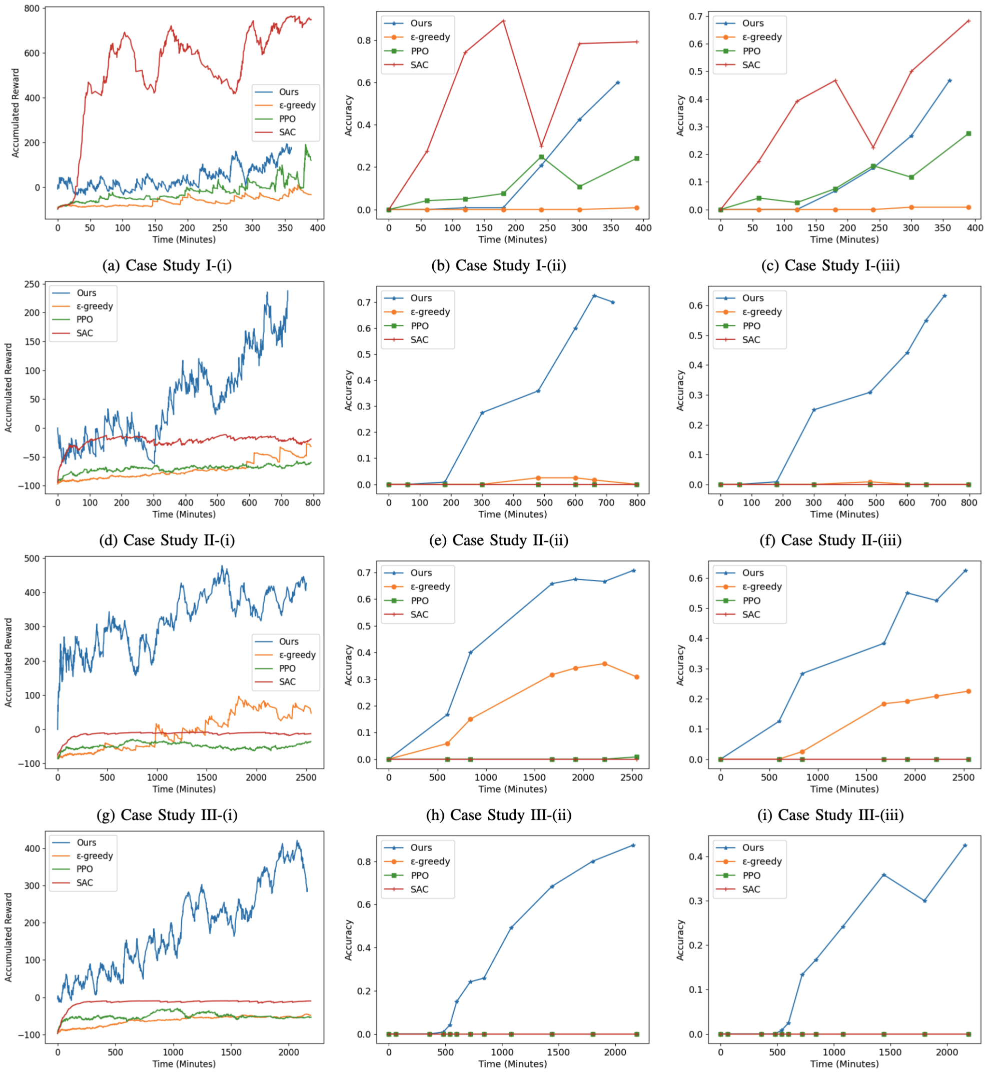
4.2. Comparative Numerical Experiments

Case Study I: Consider a navigation task where the agent must visit regions r_1, r_2, and r_3 in any order while avoiding obstacles in environment group A. This task is represented by the LTL formula $\phi = \Diamond\pi^{r_1} \wedge \Diamond\pi^{r_2} \wedge \Diamond\pi^{r_3} \wedge \Box\neg\pi^{W_{\text{obs}}}$, corresponding to a DRA with 9 states and 1 accepting pair. We set $T_g = 0.5$ hour and $T_{\epsilon,\delta} = 6$ hours. The return and the test-time accuracy in training and test environments are shown in Fig. 2(a), 2(b), and 2(c). SAC performed best across all metrics, while our method outperformed both PPO and the ϵ-greedy approach. As the training time increases, the test time accuracy of all methods tends to increase. Also, the test-time accuracy of all controllers demonstrates a decline ($\sim$10%) in unseen environments compared to the training set.

Case Study II: Next, we use the same LTL task as Case Study I but in environment group B. We have $T_g = 1.28$ hour and $T_{\epsilon,\delta} = 12$ hours. The performance of all methods is demonstrated in Fig. 2. We observe that all the baselines struggle to gather non-negative rewards or develop a satisfactory test-time policy due to the environmental complexity. The test-time accuracy of all baselines is almost 0%. Our method yields 72.5% accuracy in training and 63% in testing.

Case Study III: Next, we consider a task requiring the agent to visit known regions r_1, r_2, r_3, r_4 in any order, as long as r_4 is visited only after r_1 is visited, and always avoid obstacles in environment group A. This LTL formula is $\phi = \Diamond\pi^{r_1} \wedge \Diamond\pi^{r_4} \wedge (\neg\pi^{r_4}\mathcal{U}\pi^{r_1}) \wedge \Diamond\pi^{r_2} \wedge \Diamond\pi^{r_3} \wedge \Box\neg\pi^{W_{\text{obs}}}$ corresponding to a DRA with 13 states and 1 accepting pair. We have $T_g = 0.5$ hour and $T_{\epsilon,\delta} = 42$ hours. This harder task demands visiting more regions in a fixed sequence. Results in Figs. 2(g)-2(i) show that our method outperforms the baselines in all metrics. PPO and SAC failed to earn positive rewards during training, while DQN only began collecting non-negative rewards after 1,000 minutes. Test accuracies for PPO, SAC, and DQN were 0%, 0%, and 22.5%, respectively. In contrast, our method achieved an accuracy of 70.8% in training and 62.5% in test environments.

Case Study IV: In this long-horizon task, the agent must eventually visit r_1 before r_4, revisit r_2 and r_3 infinitely often while always avoiding $\mathcal{W}_{\text{obs}}$ in environment group A. This task is specified by



(a) Case Study I-(i)

(b) Case Study I-(ii)

(c) Case Study I-(iii)

(d) Case Study II-(i)

(e) Case Study II-(ii)

(f) Case Study II-(iii)

(g) Case Study III-(i)

(h) Case Study III-(ii)

(i) Case Study III-(iii)

(j) Case Study IV-(i)

(k) Case Study IV-(ii)

(l) Case Study IV-(iii)

Figure 2: Illustration of the evaluation metrics in each case study. Each row represents a single case study. Columns 1–3 plot metrics (i), (ii), and (iii), respectively. Line colors: Ours (blue), DQN (orange), PPO (green), and SAC (red).

the LTL formula $\phi = \Diamond\pi^{r_1} \wedge \Diamond\pi^{r_4} \wedge (\neg\pi^{r_4}\mathcal{U}\pi^{r_1}) \wedge \Box\Diamond\pi^{r_2} \wedge \Box\Diamond\pi^{r_3} \wedge \Box\neg\pi^{\mathcal{W}_{\text{obs}}}$, corresponding to a DRA with 10 states and 1 accepting pair. This task is more complex than previous ones, requiring repeated visits to specific regions. We set $T_{\epsilon,\delta} = 36$ hours and $T_g = 0.5$ hour. The performance is shown in Figs. 2(j) - 2(l). Due to the increased complexity, all baselines fail to collect positive rewards or learn effective policies, resulting in 0% test accuracy. In contrast, our method achieves high returns, with 88.3% accuracy in training and 42.5% in test environments.

5. Conclusions & Future Work

We proposed a new sample-efficient deep RL algorithm for LTL-encoded tasks. Its sample efficiency relies on prioritizing exploration in the vicinity of task-related regions, as supported by our comparative experiments. Our future work will extend the proposed framework to high-dimensional state and action spaces and evaluate it on missions beyond navigation, such as locomotion tasks.

Acknowledgments

This work was supported in part by the NSF award CNS #2231257. We also thank Haojun Chen for helping set up the hardware demonstrations.

References

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

Anand Balakrishnan, Stefan Jaksic, Edgar Aguilar, Dejan Nickovic, and Jyotirmoy Deshmukh. Model-free reinforcement learning for symbolic automata-encoded objectives. In *25th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–2, 2022.

Suguman Bansal. Specification-guided reinforcement learning. In *Static Analysis: 29th International Symposium, SAS, Auckland, New Zealand, December 5–7, 2022, Proceedings*, pages 3–9. Springer.

Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10349–10355. IEEE, 2020.

Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtěch Forejt, Jan Křetínskỳ, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114. Springer, 2014.

Mingyu Cai, Hosein Hasanbeig, Shaoping Xiao, Alessandro Abate, and Zhen Kan. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE RA-L*, 6(4): 7973–7980, 2021a.

Mingyu Cai, Shaoping Xiao, Baoluo Li, Zhiliang Li, and Zhen Kan. Reinforcement learning based temporal logic control with maximum probabilistic satisfaction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 806–812. IEEE, 2021b.

Mingyu Cai, Makai Mann, Zachary Serlin, Kevin Leahy, and Cristian-Ioan Vasile. Learning minimally-violating continuous control for infeasible linear temporal logic specifications. *arXiv preprint arXiv:2210.01162*, 2022.

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*, 8(4):2158–2165, 2023.

Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE, 2017.

Max H Cohen and Calin Belta. Model-based reinforcement learning for approximate optimal control with temporal logic specifications. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.

Daniel Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE ICRA*, pages 5113–5120, 2018.

Jie Fu and Ufuk Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. *arXiv preprint arXiv:1404.7073*, 2014.

Qitong Gao, Davood Hajinezhad, Yan Zhang, Yiannis Kantaros, and Michael M Zavlanos. Reduced variance deep reinforcement learning with temporal logic specifications. In *10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 237–248, 2019.

Meng Guo and Michael M. Zavlanos. Distributed data gathering with buffer constraints and intermittent communication. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

Meng Guo and Michael M Zavlanos. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control*, 63(12):4051–4066, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Hosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *58th IEEE Conference on Decision and Control (CDC)*, Nice, France, 2019.

Hosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–22. Springer, 2020.

Hosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. DeepSynth: Program synthesis for automatic task segmentation in deep reinforcement learning. In *AAAI*. Association for the Advancement of Artificial Intelligence, 2021.

Hosein Hasanbeig, Daniel Kroening, and Alessandro Abate. LCRL: Certified policy synthesis via logically-constrained reinforcement learning. In *CONFEST*, pages 217–231. Springer, 2022.

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*, 2022.

Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-logic-based reward shaping for continuing reinforcement learning tasks. In *Proceedings of the AAAI Conference on artificial Intelligence*, volume 35, pages 7995–8003, 2021.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34, 2021.

Krishna C Kalagarla, Rahul Jain, and Pierluigi Nuzzo. Model-free reinforcement learning for optimal control of Markov decision processes under signal temporal logic specifications. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2252–2257. IEEE, 2021.

Yiannis Kantaros. Accelerated reinforcement learning for temporal logic control objectives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyoto, Japan, October 2022.

Yiannis Kantaros and Jun Wang. Sample-efficient reinforcement learning with temporal logic objectives: Leveraging the task specification to guide exploration. *IEEE Transactions on Automatic Control*, pages 1–16, 2024. doi: 10.1109/TAC.2024.3484290.

Yiannis Kantaros and Michael M Zavlanos. Distributed intermittent connectivity control of mobile robot networks. *IEEE Transactions on Automatic Control*, 62(7):3109–3121, 2016.

Yiannis Kantaros and Michael M Zavlanos. STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research*, 39(7):812–836, 2020.

Yiannis Kantaros, Samarth Kalluraya, Qi Jin, and George J Pappas. Perception-based temporal logic planning in uncertain semantic maps. *IEEE Transactions on Robotics*, 38(4):2536–2556, 2022.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

Abolfazl Lavaei, Fabio Somenzi, Sadegh Soudjani, Ashutosh Trivedi, and Majid Zamani. Formal controller synthesis for continuous-space MDPs via model-free reinforcement learning. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 98–107. IEEE, 2020.

Kevin Leahy, Dingjiang Zhou, Cristian-Ioan Vasile, Konstantinos Oikonomopoulos, Mac Schwager, and Calin Belta. Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints. *Autonomous Robots*, 40(8):1363–1378, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Rewat Sachdeva, Raghav Gakhar, Sharad Awasthi, Kavinder Singh, Ashutosh Pandey, and Anil Singh Parihar. Uncertainty and noise aware decision making for autonomous vehicles-a bayesian approach. *IEEE Transactions on Vehicular Technology*, 2024.

Brent Schlotfeldt, Dinesh Thakur, Nikolay Atanasov, Vijay Kumar, and George J Pappas. Anytime planning for decentralized multirobot active information gathering. *IEEE Robotics and Automation Letters*, 3(2):1025–1032, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Daqian Shao and Marta Kwiatkowska. Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

Chuanzheng Wang, Yinan Li, Stephen L Smith, and Jun Liu. Continuous motion planning with temporal logic specifications using deep neural networks. *arXiv preprint arXiv:2004.02610*, 2020.

Zhe Xu and Ufuk Topcu. Transfer of temporal logic formulas in reinforcement learning. In *IJCAI: proceedings of the conference*, volume 28, page 4010. NIH Public Access, 2019.

Yuexiang Zhai, Christina Baek, Zhengyuan Zhou, Jiantao Jiao, and Yi Ma. Computational benefits of intermediate rewards for goal-reaching policy learning. *Journal of Artificial Intelligence Research*, 73:847–896, 2022.

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*, 2023.

Weichao Zhou and Wenchao Li. Programmatic reward design by example. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9233–9241, 2022.

Weitao Zhou, Zhong Cao, Nanshan Deng, Kun Jiang, and Diange Yang. Identify, estimate and bound the uncertainty of reinforcement learning for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 24(8):7932–7942, 2023.