DEEPLTL: LEARNING TO EFFICIENTLY SATISFY COMPLEX LTL SPECIFICATIONS FOR MULTI-TASK RL

Mathias Jackermeier, Alessandro Abate
Department of Computer Science, University of Oxford
{mathias.jackermeier,alessandro.abate}@cs.ox.ac.uk

ABSTRACT

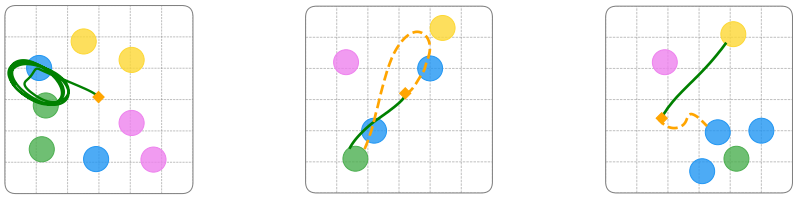
Linear temporal logic (LTL) has recently been adopted as a powerful formalism for specifying complex, temporally extended tasks in multi-task reinforcement learning (RL). However, learning policies that efficiently satisfy arbitrary specifications not observed during training remains a challenging problem. Existing approaches suffer from several shortcomings: they are often only applicable to finite-horizon fragments of LTL, are restricted to suboptimal solutions, and do not adequately handle safety constraints. In this work, we propose a novel learning approach to address these concerns. Our method leverages the structure of Büchi automata, which explicitly represent the semantics of LTL specifications, to learn policies conditioned on sequences of truth assignments that lead to satisfying the desired formulae. Experiments in a variety of discrete and continuous domains demonstrate that our approach is able to zero-shot satisfy a wide range of finite- and infinite-horizon specifications, and outperforms existing methods in terms of both satisfaction probability and efficiency. Code available at: https://deep-ltl.github.io/

1 INTRODUCTION

One of the fundamental challenges in artificial intelligence (AI) is to create agents capable of following arbitrary instructions. While significant research efforts have been devoted to designing reinforcement learning (RL) agents that can complete tasks expressed in natural language (Oh et al., 2017; Goyal et al., 2019; Luketina et al., 2019), recent years have witnessed increased interest in *formal* languages to specify tasks in RL (Andreas et al., 2017; Camacho et al., 2019; Jothimurugan et al., 2021). Formal specification languages offer several desirable properties over natural language, such as well-defined semantics and compositionality, allowing for the specification of unambiguous, structured tasks (Vaezipoor et al., 2021; León et al., 2022). Recent works have furthermore shown that it is possible to automatically translate many natural language instructions to formal languages, providing interpretable yet precise representations of tasks (Pan et al., 2023; Liu et al., 2023).

Linear temporal logic (LTL) (Pnueli, 1977) in particular has been adopted as a powerful formalism for instructing RL agents (Hasanbeig et al., 2018; Araki et al., 2021; Voloshin et al., 2023). LTL is an appealing specification language that allows for the definition of tasks in terms of high-level features of the environment. These tasks can utilise complex compositional structure, are inherently temporally extended (i.e. non-Markovian), and naturally incorporate safety constraints.

While several approaches have been proposed to learning policies capable of zero-shot executing arbitrary LTL specifications in a multi-task RL setting (Kuo et al., 2020; Vaezipoor et al., 2021; Qiu et al., 2023; Liu et al., 2024), they suffer from several limitations. First, most existing methods are limited to subsets of LTL and cannot handle infinite-horizon (i.e. ω-regular) specifications, which form an important class of objectives including *persistence* (eventually, a desired state needs to be reached forever), *recurrence* (a set of states needs to be reached infinitely often), and *response* (whenever a particular event happens, a task needs to be completed) (Manna & Pnueli, 1990). Second, many current techniques are *myopic*, that is, they solve tasks by independently completing individual subtasks, which can lead to inefficient, globally suboptimal solutions (Vaezipoor et al., 2021). Finally, existing approaches often do not adequately handle safety constraints of specifications, especially when tasks can be completed in multiple ways with different safety implications. For an illustration of these limitations, see Figure 1.



(a) ω-regular tasks (b) Optimality (c) Safety

Figure 1: Limitations of existing methods, illustrated via trajectories in the *ZoneEnv* environment. The initial agent position is denoted as an orange diamond. (a) Most existing approaches cannot handle infinite-horizon tasks, such as $\mathsf{G}\,\mathsf{F}\,\mathsf{blue} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{green}$. (b) Given the formula $\mathsf{F}\,(\mathsf{blue} \wedge \mathsf{F}\,\mathsf{green})$, a *myopic* approach produces a suboptimal solution (orange line). We prefer the more efficient green trajectory. (c) Given the task $(\mathsf{F}\,\mathsf{green} \vee \mathsf{F}\,\mathsf{yellow}) \wedge \mathsf{G}\,\neg\mathsf{blue}$, the agent should aim to reach the yellow region instead of the green region, since this is the safer option. Many existing approaches are unable to perform this sort of planning.

In this paper, we develop a novel approach to learning policies for zero-shot execution of LTL specifications that addresses these shortcomings. Our method exploits the structure of Büchi automata to non-myopically reason about ways of completing a (possibly infinite-horizon) specification, and to ensure that safety constraints are satisfied. Our main contributions are as follows:

- we develop the *first* non-myopic approach to learning multi-task policies for zero-shot execution of LTL specifications that is applicable to infinite-horizon tasks;
- we propose a novel representation of LTL formulae based on reach-avoid sequences of truth assignments, which allows us to learn policies that intrinsically consider safety constraints;
- we propose a novel neural network architecture that combines DeepSets and RNNs to condition the policy on the desired specification;
- lastly, we empirically validate the effectiveness of our method on a range of environments and tasks, demonstrating that it outperforms existing approaches in terms of satisfaction probability and efficiency.

2 Background

Reinforcement learning. We model RL environments using the framework of *Markov decision processes* (MDPs). An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the *unknown* transition kernel, $\mu \in \Delta(\mathcal{S})$ is the initial state distribution, $r\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

We denote the probability of transitioning from state s to state s' after taking action a as $\mathcal{P}(s' \mid s, a)$. A (memoryless) *policy* $\pi\colon \mathcal{S} \to \Delta(\mathcal{A})$ is a map from states to probability distributions over actions. Executing a policy π in an MDP gives rise to a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$, where $s_0 \sim \mu$, $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$. The goal of RL is to find a policy π^* that maximises the *expected discounted return* $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where we write $\tau \sim \pi$ to indicate that the distribution over trajectories depends on the policy π. The *value function* of a policy $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$ is defined as the expected discounted return starting from state s and following policy π thereafter.

Linear temporal logic. Linear temporal logic (LTL) (Pnueli, 1977) provides a formalism to precisely specify properties of infinite trajectories. LTL formulae are defined over a set of atomic propositions AP, which describe high-level features of the environment. The syntax of LTL formulae is recursively defined as

$$\mathsf{true} \mid a \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathsf{X}\,\varphi \mid \varphi\ \mathsf{U}\ \psi$$

where $a \in AP$ and φ and ψ are themselves LTL formulae. $\wedge$ and $\neg$ are the Boolean operators conjunction and negation, which allow for the definition of all standard logical connectives. The temporal operators X and U intuitively mean "next" and "until". We define the two temporal modalities F ("eventually") and G ("always") as $\mathsf{F}\,\varphi \equiv \mathsf{true}\ \mathsf{U}\ \varphi$ and $\mathsf{G}\,\varphi \equiv \neg\mathsf{F}\,\neg\varphi$.

The semantics of LTL align with the intuitive meanings of its operators. For example, in the *ZoneEnv* environment depicted in Figure 1, the atomic propositions AP correspond to coloured regions. We can naturally express many desirable tasks as LTL specifications, such as reaching a blue region (F blue), avoiding blue until a yellow region is reached ($\neg$blue U yellow), reaching and remaining in a green region ($\mathsf{F}\,\mathsf{G}$ green), or oscillating between blue and green regions while avoiding yellow ($\mathsf{G}\,\mathsf{F}$ green $\wedge$ $\mathsf{G}\,\mathsf{F}$ blue $\wedge$ $\mathsf{G}\,\neg$yellow). The latter two examples represent *infinite-horizon* specifications, which describe behaviour over an infinite time horizon.

Formally, the satisfaction semantics of LTL are defined via a recursive satisfaction relation $w \models \varphi$ on infinite sequences w of truth assignments[1] over AP (i.e. ω-words over 2^{AP}) (see Appendix A for details). To ground LTL specifications in an MDP, we assume access to a *labelling function* $L\colon \mathcal{S} \to 2^{AP}$, which returns the atomic propositions that are true in a given state. A trajectory τ is mapped to a sequence of assignments via its trace $\mathrm{Tr}(\tau) = L(s_0)L(s_1)\ldots$, and we write $\tau \models \varphi$ as shorthand for $\mathrm{Tr}(\tau) \models \varphi$. The *satisfaction probability* of an LTL formula φ under policy π is then defined as $\Pr(\pi \models \varphi) = \mathbb{E}_{\tau\sim\pi}\big[\mathbb{1}[\tau \models \varphi]\big]$.

Büchi automata. A more practical way of reasoning about the semantics of LTL formulae is via *Büchi automata* (Büchi, 1966), which are specialised automata that can be constructed to keep track of the progress towards satisfying a specification. In particular, in this work we focus on *limit-deterministic Büchi automata* (LDBAs) (Sickert et al., 2016), which are particularly amenable to be employed with MDPs. An LDBA is a tuple $\mathcal{B} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E})$, where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{AP}$ is a finite alphabet, $\delta\colon \mathcal{Q} \times (\Sigma \cup \mathcal{E}) \to \mathcal{Q}$ is the transition function, and $\mathcal{F}$ is the set of accepting states. Additionally, $\mathcal{Q}$ is composed of two disjoint subsets $\mathcal{Q} = \mathcal{Q}_N \uplus \mathcal{Q}_D$ such that $\mathcal{F} \subseteq \mathcal{Q}_D$ and $\delta(q, \alpha) \in \mathcal{Q}_D$ for all $q \in \mathcal{Q}_D$ and $\alpha \in \Sigma$. The set $\mathcal{E}$ is an indexed set of ε-transitions (a.k.a jump transitions), which transition from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ without consuming any input, and there are no other transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$.

A *run* r of $\mathcal{B}$ on the ω-word w is an infinite sequence of states in $\mathcal{Q}$ respecting the transition function. An infinite word w is *accepted* by $\mathcal{B}$ if there exists a run r of $\mathcal{B}$ on w that visits an accepting state infinitely often. For every LTL formula φ, we can construct an LDBA $\mathcal{B}_\varphi$ that accepts exactly the words satisfying φ (Sickert et al., 2016).

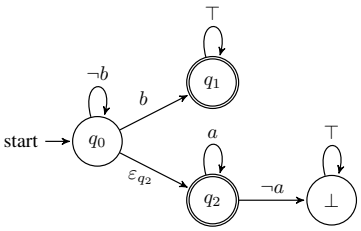


Figure 2: LDBA for the formula $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,\mathsf{b}$.

Example 1. Figure 2 depicts an LDBA for the formula $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,\mathsf{b}$. The automaton starts in state q_0 and transitions to the accepting state q_1 upon observing the proposition b. Once it has reached q_1, it stays there indefinitely. Alternatively, it can transition to the accepting state q_2 without consuming any input via the ε-transition. Once in q_2, the automaton accepts exactly the words where a is true at every step. □

3 Problem Setting

Our high-level goal is to find a specification-conditioned policy $\pi|\varphi$ that maximises the probability of satisfying arbitrary LTL formulae φ. Formally, we introduce an arbitrary distribution ξ over LTL specifications φ, and aim to compute the optimal policy

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

We now introduce the necessary formalism to find solutions to Equation 1 via reinforcement learning.

Definition 1 (Product MDP)**.** The *product MDP* $\mathcal{M}^\varphi$ of an MDP $\mathcal{M}$ and an LDBA $\mathcal{B}_\varphi$ synchronises the execution of $\mathcal{M}$ and $\mathcal{B}_\varphi$. Formally, $\mathcal{M}^\varphi$ has the state space $\mathcal{S}^\varphi = \mathcal{S} \times \mathcal{Q}$, action space $\mathcal{A}^\varphi = \mathcal{A} \times \mathcal{E}$, initial state distribution $\mu^\varphi(s, q) = \mu(s) \cdot \mathbb{1}[q = q_0]$, and transition function

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

[1] An *assignment* a is a subset of AP. Propositions $p \in a$ are assigned *true*, whereas $p \notin a$ are assigned *false*.

The product MDP $\mathcal{M}^\varphi$ extends the state space of $\mathcal{M}$ in order to keep track of the current state of the LDBA. This allows us to consider only *memoryless* policies that map tuples (s, q) of MDP and LDBA states to actions, since the LDBA takes care of adding the memory necessary to satisfy φ (Baier & Katoen, 2008). Quite importantly, note that in practice we never build the product MDP explicitly, but instead simply update the current LDBA state q with the propositions $L(s)$ observed at each time step. Also note that the action space in $\mathcal{M}^\varphi$ is extended with $\mathcal{E}$ to allow the policy to follow ε-transitions in $\mathcal{B}_\varphi$ without acting in the MDP. Trajectories in $\mathcal{M}^\varphi$ are sequences $\tau = ((s_0, q_0), a_0, (s_1, q_1), a_1, \ldots)$, and we denote as τ_q the projection of τ onto the LDBA states $q_0, q_1, \ldots$. We can restate the satisfaction probability of formula φ in $\mathcal{M}^\varphi$ as

$$\Pr(\pi \models \varphi) = \mathop{\mathbb{E}}_{\tau\sim\pi}\big[\mathbb{1}\left[\inf(\tau_q) \cap \mathcal{F} \neq \emptyset\right]\big],$$

where $\inf(\tau_q)$ denotes the set of states that occur infinitely often in τ_q.

An optimal policy can be found via goal-conditioned reinforcement learning in the product MDP as follows: given some probability distribution ξ over LTL formulae, sample $\varphi \sim \xi$ and trajectories $\tau \sim \pi|\varphi$, assigning a positive reward of 1 whenever the policy visits an accepting state in the LDBA corresponding to φ, and 0 otherwise. To ensure that the reward-maximising policy π^*_{RL} indeed maximises Equation 1, one can employ *eventual discounting* (Voloshin et al., 2023) and only discount visits to accepting states in the automaton and not the steps between those visits (for a further discussion, see Appendix B.1). With this, we obtain the following bound on the performance of π^*_{RL}:

Theorem 1 (Corollary of Voloshin et al. (2023), Theorem 4.2)**.** *For any $\gamma \in (0, 1)$ we have*

$$\sup_\pi \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi \models \varphi)] - \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi^*_{RL} \models \varphi)] \leq 2\log(\frac{1}{\gamma}) \sup_\pi O_\pi,$$

where $O_\pi = \mathbb{E}_{\varphi\sim\xi,\tau\sim\pi|\varphi}\big[|\{q \in \tau_q : q \in \mathcal{F}_{\mathcal{B}_\varphi}\}| \,\big|\, \tau \not\models \varphi\big]$ is the expected number of visits to accepting states for trajectories that do not satisfy a specification.

Proof. The proof follows from (Voloshin et al., 2023, Theorem 4.2) by repeated application of the linearity of expectation and triangle inequality. A detailed proof is given in Appendix B.2. □

However, while eventual discounting provides desirable theoretical guarantees, it does not take into account any notion of *efficiency* of formula satisfaction, which is an important practical concern. Consider for example the simple formula $\mathsf{F}\,\mathsf{a}$. Eventual discounting assigns the same return to all policies that eventually visit s with $\mathsf{a} \in L(s)$, regardless of the number of steps required to materialise a. In practice, we often prefer policies that are more efficient (require fewer steps to make progress towards satisfaction), even if their overall satisfaction probability might be slightly reduced. A natural way to formalise this tradeoff is as follows:

Problem 1 (Efficient LTL satisfaction)**.** Given an unknown MDP $\mathcal{M}$, a distribution over LTL formulae ξ, and LDBAs $\mathcal{B}_\varphi$ for each $\varphi \in \mathrm{supp}(\xi)$ (the support of ξ), find the optimal policy

$$\pi^* = \arg\max_\pi \mathop{\mathbb{E}}_{\substack{\varphi\sim\xi,\\ \tau\sim\pi|\varphi}} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[q_t \in \mathcal{F}_{\mathcal{B}_\varphi}]\right]. \tag{2}$$

Here, we discount *all* time steps, such that more efficient policies receive higher returns. While solutions to Problem 1 are not guaranteed to be probability-optimal (as per Equation 1), they will generally still achieve a high probability of formula satisfaction, while also taking efficiency into account. We consider Problem 1 for the rest of this paper, since we believe efficiency to be an important practical concern, but note that our approach is equally applicable to the eventual discounting setting (see Appendix B.3). There is an interesting connection to *discounted LTL* (Almagor et al., 2014; Alur et al., 2023) (see Appendix C for details), but we leave a full investigation thereof for future work.

4 METHOD

Solving Problem 1 requires us to train a policy conditioned on the current MDP state s and the current state q of an LDBA constructed from a given target specification φ. Our key insight is that we can extract a useful representation of q directly from the structure of the LDBA, by reasoning about the possible ways of satisfying the given formula from the current LDBA state q. This representation is then used to condition the policy, and guide the agent towards satisfying a given specification.

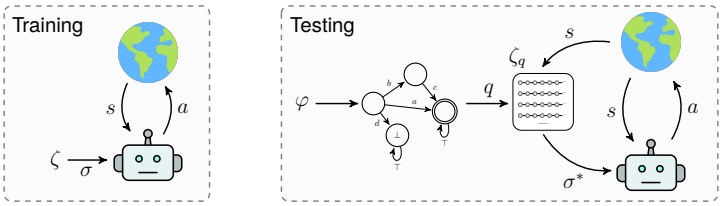


Figure 3: Overview of our approach. (*Left*) During training, we train a general sequence-conditioned policy with reach-avoid sequences σ. (*Right*) At test time, we construct an LDBA from the target specification φ. We then select the optimal reach-avoid sequence σ^* for the current LDBA state q according to the value function $V^\pi(s, \sigma)$, and produce an action a from the policy conditioned on σ^*.

4.1 Representing LTL specifications as sequences

Computing accepting cycles. An optimal policy for Problem 1 must continuously visit accepting states in $\mathcal{B}_\varphi$. Since $\mathcal{B}_\varphi$ is finite, this means that the agent has to reach an *accepting cycle* in the LDBA. Intuitively, the possible ways of reaching accepting cycles are an informative representation of the current LDBA state q, as they capture how to satisfy the given task. We compute all possible ways of reaching an accepting cycle using an algorithm based on depth-first search (DFS) that explores all possible paths from q to an accepting state $q_f \in \mathcal{F}$, and then back to a state already contained in the path (see Appendix E for details). Let P_q denote the resulting set of paths from q to accepting cycles.

Remark. In the case that φ corresponds to a task that can be completed in finite time (e.g. $\mathsf{F}\,\mathsf{a}$), the accepting cycle in $\mathcal{B}_\varphi$ is trivial and consists of only a single looping state (see e.g. q_1 in Figure 2). We note that our algorithm to compute accepting cycles is similar to previous work on reward shaping (Shah et al., 2024); however, we use the accepting cycles for an altogether different purpose.

From paths to sequences. A path $p \in P_q$ is an infinite sequence of states $(q_1, q_2, \ldots)$ in the LDBA. Let $A_i^+ = \{a : \delta(q_i, a) = q_{i+1}\}$ denote the set of assignments $a \in 2^{AP}$ that progress the LDBA from state q_i to q_{i+1}.[2] We furthermore define the set of negative assignments $A_i^- = \{a : a \notin A_i^+ \wedge \delta(q_i, a) \neq q_i\}$ that lead from q_i to a different state in the LDBA (excluding self-loops). In order to satisfy the LTL specification via p, the policy has to sequentially visit MDP states s_t such that $L(s_{t_i}) \in A_i^+$ for some t_i, while avoiding assignments in A_i^-. We refer to the sequence

$$\sigma_p = \big((A_1^+, A_1^-), (A_2^+, A_2^-), \ldots\big)$$

as the *reach-avoid sequence* corresponding to p, and denote as $\zeta_q = \{\sigma_p : p \in P_q\}$ the set of all reach-avoid sequences for q.

Example 2. The first two steps of $\sigma = \big((\{\{\mathsf{a}\}\}, \{\{\mathsf{b}, \mathsf{d}\}\}), (\{\{\mathsf{c}\}, \{\mathsf{e}\}\}, \emptyset), \ldots\big)$ require the agent to achieve a while avoiding states with both b and d, and subsequently achieve c or e. □

4.2 Overview of the approach

See Figure 3 for an overview of our method. Representing the current LDBA state q as a set of reach-avoid sequences allows us to condition the policy on possible ways of achieving the given specification. On a high level, our approach works as follows: in the *training stage*, we learn a general sequence-conditioned policy $\pi\colon \mathcal{S} \times \zeta \to \Delta(\mathcal{A})$ together with its value function $V^\pi\colon \mathcal{S} \times \zeta \to \mathbb{R}$ to satisfy arbitrary reach-avoid sequences $\sigma \in \zeta$ over AP, following the standard framework of *goal-conditioned* RL (Liu et al., 2022). Note that we do not assume access to a distribution ξ over formulae, since we are interested in satisfying arbitrary specifications. At *test time*, we are now given a target specification φ and construct its corresponding LDBA. We keep track of the current LDBA state q, and select the optimal reach-avoid sequence to follow in order to satisfy φ according to the value function of π, i.e.

$$\sigma^* = \arg\max_{\sigma \in \zeta_q} V^\pi(s, \sigma). \tag{3}$$

We then execute actions $a \sim \pi(\cdot, \sigma^*)$ until the next LDBA state is reached.

[2]For now, we assume that there are no ε-transitions in p. We revisit ε-transitions in Section 4.5.

$$\sigma = \left((A_1^+, A_1^-), (A_2^+, A_2^-), \dots\right) \xrightarrow{\textit{DeepSets}} \left(\boldsymbol{e}_{A_1^+} \| \boldsymbol{e}_{A_1^-},\ \boldsymbol{e}_{A_2^+} \| \boldsymbol{e}_{A_2^-}, \dots\right) \xrightarrow{\textit{RNN}} \boldsymbol{e}_\sigma$$

Figure 4: Illustration of the *sequence module*. The positive and negative assignments in the truncated reach-avoid sequence σ are encoded using the *DeepSets* architecture, which produces embeddings $\boldsymbol{e}_A$. These are then concatenated and passed through an RNN, which yields the final sequence representation $\boldsymbol{e}_\sigma$.

The test-time execution of our approach can be equivalently thought of as executing a policy $\widetilde{\pi}$ in the product MDP $\mathcal{M}^\varphi$, where $\widetilde{\pi}(s, q) = \pi(s, \sigma^*)$. That is, $\widetilde{\pi}$ exploits π to reach an accepting cycle in the LDBA of the target specification, and thus approximates Problem 1. Next, we describe the model architecture of the sequence-conditioned policy, and give a detailed description of the training procedure and test-time execution.

4.3 Model Architecture

We parameterise the sequence-conditioned policy π using a modular neural network architecture. This consists of an *observation module*, which processes observations from the environment, a *sequence module*, which encodes the reach-avoid sequence, and an *actor module*, which takes as input the features produced by the previous two modules and outputs a distribution over actions.

The *observation module* is implemented as either a fully-connected (for generic state features) or convolutional neural network (for image-like observations). The *actor module* is another fully connected neural network that outputs the mean and standard deviation of a Gaussian distribution (for continuous action spaces) or the parameters of a categorical distribution (in the discrete setting). Finally, the *sequence module* consists of a permutation-invariant neural network that encodes sets of assignments, and a recurrent neural network (RNN) that maps the resulting sequence to a final representation. We discuss these components in further detail below and provide an illustration of the sequence module in Figure 4.

Representing sets of assignments. The first step of the sequence module consists in encoding the sets of assignments in a reach-avoid sequence. We employ the *DeepSets* architecture (Zaheer et al., 2017) to obtain an encoding $\boldsymbol{e}_A$ of a set of assignments A. That is, we have

$$\boldsymbol{e}_A = \rho\left(\sum_{a \in A} \phi(a)\right), \tag{4}$$

where $\phi(a)$ is a learned embedding function, and ρ is a learned non-linear transformation. Note that the resulting encoding $\boldsymbol{e}_A$ is *permutation-invariant*, i.e. it does not depend on the order in which the elements in A are processed, and Equation 4 is thus a well-defined function on sets.

Representing reach-avoid sequences. Once we have obtained encodings of the sets A_i^+ and A_i^- for each element in the reach-avoid sequence σ, we concatenate these embeddings and pass them through an RNN to yield the final representation of the sequence. Since σ is an infinite sequence, we approximate it with a finite prefix by repeating its looping part k times, such that the truncated sequence visits an accepting state exactly k times. We apply the RNN backwards, that is, from the end of the truncated sequence to the beginning, since earlier elements in σ are more important for the immediate actions of the policy. The particular model of RNN we employ is a *gated recurrent unit* (GRU) (Cho et al., 2014).

4.4 Training Procedure

We train the policy π and the value function V^π using the general framework of *goal-conditioned RL* (Liu et al., 2022). That is, we generate a random reach-avoid sequence at the beginning of each training episode and reward the agent for successfully completing it. In particular, given a training sequence $\sigma = \left((A_1^+, A_1^-), \dots, (A_n^+, A_n^-)\right)$, we keep track of the task satisfaction progress via an index $i \in [n]$ (where initially $i = 1$). We say the agent *satisfies* a set of assignments A at time step t if $L(s_t) \in A$. Whenever the agent satisfies A_i^+, we increment i by one. If $i = n + 1$, we

give the agent a reward of 1 and terminate the episode. If the agent at any point satisfies A_i^-, we also terminate the episode and give it a negative reward of -1. Otherwise, the agent receives zero reward. By maximising the expected discounted return, the policy learns to efficiently satisfy arbitrary reach-avoid sequences. In our experiments, we use *proximal policy optimisation* (PPO) (Schulman et al., 2017) to optimise the policy, but our approach can be combined with any RL algorithm.

Curriculum learning. To improve the training of π in practice, we employ a simple form of *curriculum learning* (Narvekar et al., 2020) in order to gradually expose the policy to more challenging tasks. A curriculum consists of multiple stages that correspond to training sequences of increasing length and complexity. For example, the first stage generally consists only of simple reach-tasks of the form $\sigma = \big((\{\{p\}\}, \emptyset)\big)$ for $p \in AP$, while later stages involve longer sequences with avoidance conditions. Once the policy achieves satisfactory performance on the current tasks, we move on to the next stage. For details on the exact curricula we use in our experiments, see Appendix F.4.

4.5 Test time policy execution

At test time, we execute the trained sequence-conditioned policy π to complete an arbitrary task φ. As described in Section 4.2, we keep track of the current LDBA state q in $\mathcal{B}_\varphi$, and use the learned value function V^π to select the optimal reach-avoid sequence σ^* to follow from q in order to satisfy φ (Equation 3). Note that it follows from the reward of our training procedure that

$$V^\pi(s, \sigma) \leq \mathop{\mathbb{E}}_{\tau \sim \pi | \sigma} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[i = n+1] \,\middle|\, s_0 = s \right],$$

i.e. the value function is a lower bound of the discounted probability of reaching an accepting state k times via σ (where k is the number of loops in the truncated sequence). As $k \to \infty$, the sequence σ^* that maximises V^π thus maximises a lower bound on Problem 1 for the trained policy π. Once σ^* has been selected, we execute actions $a \sim \pi(\cdot, \sigma^*)$ until the next LDBA state is reached.

Strict negative assignments. Recall that a negative assignment in a reach-avoid sequence σ_p is any assignment that leads to an LDBA state other than the desired next state in p. In practice, we find that trying to avoid all other states in the LDBA can be too restrictive for the policy. We therefore only regard as negative those assignments that lead to a significant reduction in expected performance. In particular, given a threshold λ, we define the set of *strict* negative assignments for state $q_i \in p$ as the assignments that lead to a state q' where

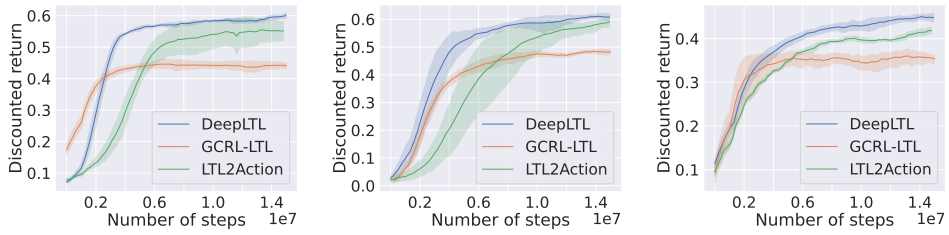
$$V^\pi(s, \sigma_p[i \ldots]) - \max_{\sigma' \in \zeta_{q'}} V^\pi(s, \sigma') \geq \lambda.$$

We then set A_i^- to be the set of *strict* negative assignments for q_i. Reducing λ leads to a policy that more closely follows the selected path p, whereas increasing λ gives the policy more flexibility to deviate from the chosen path.

Handling ε-transitions. We now discuss how to handle ε-transitions in the LDBA. As described in Section 2, whenever the LDBA is in a state q with an ε-transition to q', the policy can choose to either stay in q or transition to q' without acting in the MDP. If the sequence σ^* chosen at q starts with an ε-transition (i.e. $A_1^+ = \{\varepsilon\}$), we extend the action space of π to include the action ε. If $\mathcal{A}$ is discrete, we simply add an additional dimension to the action space. In the continuous case, we learn the probability p of taking the ε-action and model $\pi(\cdot | s, \sigma^*)$ as a mixed continuous/discrete probability distribution (see e.g. (Shynk, 2012, Ch. 3.6)). Whenever the policy executes the ε-action, we update the current LDBA state to the next state in the selected path. In practice, we additionally only allow ε-actions if $L(s) \notin A_2^-$, since in that case taking the ε-transition would immediately lead to failure.

4.6 Discussion

We argue that our approach has several advantages over existing methods. Since we operate on accepting cycles of Büchi automata, our method is applicable to infinite-horizon (i.e. ω-regular) tasks, contrary to most existing approaches. Our method is the *first* approach that is also non-myopic, as it is able to reason about the entire structure of a specification via temporally extended reach-avoid



(a) LetterWorld (b) ZoneEnv (c) FlatWorld

Figure 5: Evaluation curves on *reach/avoid* specifications. Each datapoint is collected by averaging the discounted return of the policy across 50 episodes with randomly sampled tasks, and shaded areas indicate 90% confidence intervals over 5 different random seeds.

sequences. This reasoning naturally considers safety constraints, which are represented through negative assignments and inform the policy about which propositions to avoid. Crucially, these safety constraints are considered during planning, i.e. when selecting the optimal sequence to execute, rather than only during execution. For a detailed comparison of our approach to related work, see Section 6.

5 Experiments

We evaluate our approach, called *DeepLTL*, in a variety of environments and on a range of LTL specifications of varying difficulty. We aim to answer the following questions: **(1)** Is DeepLTL able to learn policies that can zero-shot satisfy complex LTL specifications? **(2)** How does our method compare to relevant baselines in terms of both satisfaction probability and efficiency? **(3)** Can our approach successfully handle infinite-horizon specifications?

5.1 Experimental setup

Environments. Our experiments involve different domains with varying state and action spaces. This includes the *LetterWorld* environment (Vaezipoor et al., 2021), a 7×7 discrete grid world in which letters corresponding to atomic propositions occupy randomly sampled positions in the grid. We also consider the high-dimensional *ZoneEnv* environment from Vaezipoor et al. (2021), in which a robotic agent with a continuous action space has to navigate between different randomly placed coloured regions, which correspond to the atomic propositions. Finally, we evaluate our approach on the continuous *FlatWorld* environment (Voloshin et al., 2023), in which multiple propositions can hold true at the same time. We provide further details and visualisations in Appendix F.1.

LTL specifications. We consider a range of tasks of varying complexity. *Reach/avoid* specifications are randomly sampled from a task space that encompasses both sequential reachability objectives of the form $\mathsf{F}\,(p_1 \wedge (\mathsf{F}\,p_2 \wedge (\mathsf{F}\,p_3))$ and reach-avoid tasks $\neg p_1 \;\mathsf{U}\; (p_2 \wedge (\neg p_3 \;\mathsf{U}\; p_4))$, where the p_i are randomly sampled atomic propositions. *Complex* specifications are given by more complicated, environment-specific LTL formulae, such as the specification $((\mathsf{a} \vee \mathsf{b} \vee \mathsf{c} \vee \mathsf{d}) \Rightarrow \mathsf{F}\,(\mathsf{e} \wedge (\mathsf{F}\,(\mathsf{f} \wedge \mathsf{F}\,\mathsf{g})))) \;\mathsf{U}\; (\mathsf{h} \wedge \mathsf{F}\,\mathsf{i})$ in *LetterWorld*. We also separately investigate *infinite-horizon* tasks such as $\mathsf{G}\,\mathsf{F}\,\mathsf{a} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{b}$ and $\mathsf{F}\,\mathsf{G}\,\mathsf{a}$. The specifications we consider cover a wide range of LTL objectives, including reachability, safety, recurrence, persistence, and combinations thereof. Details on the exact specifications we use in each environment are given in Appendix F.2.

Baselines. We compare DeepLTL to two state-of-the-art approaches for learning general LTL-satisfying policies. LTL2Action (Vaezipoor et al., 2021) encodes the syntax tree of a target formula via a graph neural network (GNN) and uses a procedure known as *LTL progression* to progress through the specification based on the observed propositions. The second baseline, GCRL-LTL (Qiu et al., 2023), instead learns proposition-conditioned policies and combines them compositionally using a weighted graph search on the Büchi automaton of a target specification.

Evaluation protocol. In line with previous work, the methods are trained for 15M interaction steps on each environment with PPO (Schulman et al., 2017). Details about hyperparameters and

Table 1: Evaluation results of trained policies on *complex* finite-horizon specifications. We report the *success rate* (SR) and average number of steps to satisfy the task (μ). Results are averaged over 5 seeds and 500 episodes per seed. "$\pm$" indicates the standard deviation over seeds.

		LTL2Action		GCRL-LTL		DeepLTL	
		SR (↑)	μ (↓)	SR (↑)	μ (↓)	SR (↑)	μ (↓)
LetterWorld	φ_1	$0.75_{\pm 0.18}$	$29.48_{\pm 3.20}$	$0.94_{\pm 0.05}$	$15.29_{\pm 0.70}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{9.66}_{\pm 0.35}$
	φ_2	$0.79_{\pm 0.10}$	$19.04_{\pm 6.79}$	$0.94_{\pm 0.03}$	$9.77_{\pm 1.16}$	$\mathbf{0.98}_{\pm 0.00}$	$\mathbf{7.26}_{\pm 0.35}$
	φ_3	$0.41_{\pm 0.14}$	$40.31_{\pm 2.88}$	$\mathbf{1.00}_{\pm 0.00}$	$20.72_{\pm 1.34}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{12.23}_{\pm 0.58}$
	φ_4	$0.72_{\pm 0.17}$	$28.83_{\pm 4.47}$	$0.82_{\pm 0.07}$	$14.60_{\pm 1.63}$	$\mathbf{0.97}_{\pm 0.01}$	$\mathbf{12.13}_{\pm 0.58}$
	φ_5	$0.44_{\pm 0.26}$	$31.84_{\pm 9.06}$	$\mathbf{1.00}_{\pm 0.00}$	$25.63_{\pm 0.55}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{9.48}_{\pm 0.78}$
ZoneEnv	φ_6	$0.60_{\pm 0.20}$	$424.07_{\pm 14.95}$	$0.85_{\pm 0.03}$	$452.19_{\pm 15.59}$	$\mathbf{0.92}_{\pm 0.06}$	$\mathbf{303.38}_{\pm 19.43}$
	φ_7	$0.14_{\pm 0.18}$	$416.78_{\pm 66.38}$	$0.85_{\pm 0.05}$	$451.18_{\pm 04.91}$	$\mathbf{0.91}_{\pm 0.03}$	$\mathbf{299.95}_{\pm 09.47}$
	φ_8	$0.67_{\pm 0.26}$	$414.48_{\pm 68.52}$	$0.89_{\pm 0.04}$	$449.70_{\pm 16.82}$	$\mathbf{0.96}_{\pm 0.04}$	$\mathbf{259.75}_{\pm 08.07}$
	φ_9	$0.69_{\pm 0.22}$	$331.55_{\pm 41.40}$	$0.87_{\pm 0.02}$	$303.13_{\pm 05.83}$	$\mathbf{0.90}_{\pm 0.03}$	$\mathbf{203.36}_{\pm 14.97}$
	φ_{10}	$0.66_{\pm 0.19}$	$293.22_{\pm 63.94}$	$0.85_{\pm 0.02}$	$290.73_{\pm 17.39}$	$\mathbf{0.91}_{\pm 0.02}$	$\mathbf{187.13}_{\pm 10.61}$
	φ_{11}	$0.93_{\pm 0.07}$	$123.89_{\pm 07.30}$	$0.89_{\pm 0.01}$	$137.42_{\pm 08.30}$	$\mathbf{0.98}_{\pm 0.01}$	$\mathbf{106.21}_{\pm 07.88}$
FlatWorld	φ_{12}	$\mathbf{1.00}_{\pm 0.00}$	$83.32_{\pm 01.57}$	$0.82_{\pm 0.41}$	$\mathbf{78.21}_{\pm 08.98}$	$\mathbf{1.00}_{\pm 0.00}$	$79.69_{\pm 02.50}$
	φ_{13}	$0.63_{\pm 0.50}$	$94.43_{\pm 39.30}$	$0.00_{\pm 0.00}$	$0.00_{\pm 00.00}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{52.82}_{\pm 03.09}$
	φ_{14}	$0.71_{\pm 0.40}$	$96.16_{\pm 28.93}$	$0.73_{\pm 0.41}$	$74.60_{\pm 01.86}$	$\mathbf{0.98}_{\pm 0.01}$	$\mathbf{71.76}_{\pm 02.87}$
	φ_{15}	$0.07_{\pm 0.02}$	$\mathbf{32.37}_{\pm 01.63}$	$0.73_{\pm 0.03}$	$41.30_{\pm 01.24}$	$\mathbf{0.86}_{\pm 0.01}$	$43.87_{\pm 01.45}$
	φ_{16}	$0.56_{\pm 0.35}$	$48.85_{\pm 32.85}$	$0.64_{\pm 0.08}$	$\mathbf{17.76}_{\pm 01.63}$	$\mathbf{1.00}_{\pm 0.01}$	$37.04_{\pm 05.28}$

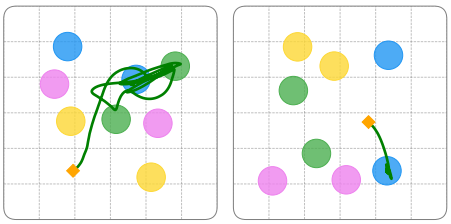
neural network architectures can be found in Appendix F.3. We report the performance in terms of discounted return over the number of environment interactions (following Equation 2) on randomly sampled *reach/avoid* tasks, and provide tabular results detailing the success rate (SR) and average number of steps until completion (μ) of trained policies on *complex* tasks. All results are averaged across 5 different random seeds. Furthermore, we provide visualisations of trajectories of trained policies for various specifications in the *ZoneEnv* and *FlatWorld* environments in Appendix G.9.

5.2 Results

Finite-horizon tasks. Figure 5 shows the discounted return achieved on *reach/avoid* tasks across environment interactions. DeepLTL clearly outperforms the baselines, both in terms of sample efficiency and final performance. The results in Table 1 further demonstrate that our method can efficiently zero-shot satisfy *complex* specifications (see Appendix F.2 for details on the tasks), achieving higher success rates (SR) than existing approaches, while requiring significantly fewer steps (μ). These results highlight the performance benefits of our representation based on reach-avoid sequences over existing encoding schemes, and show that our approach learns much more efficient policies than the myopic baseline GCRL-LTL. The higher success rates of our method furthermore indicate that it handles safety constraints better than the baselines.

Infinite-horizon tasks. Figure 6 shows example trajectories of DeepLTL on infinite-horizon tasks in the *ZoneEnv* environment. In Figure 6c we furthermore compare the performance of our approach on recurrence, i.e. $\mathsf{G}\,\mathsf{F}$, tasks (see Appendix F.2 for details) to GCRL-LTL, the only previous approach that can handle infinite-horizon specifications. We report the average number of visits to accepting states per episode, which corresponds to the number of completed cycles of target propositions (e.g. the number of times both blue and green have been visited for the specification $\mathsf{G}\,\mathsf{F}\,\mathsf{blue} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{green}$). Additional results on $\mathsf{F}\,\mathsf{G}$ tasks can be found in Appendix G.1. Our evaluation confirms that DeepLTL can successfully handle ω-regular tasks, and significantly outperforms the only relevant baseline.

Further experimental results. We provide further experimental results in Appendix G. These include more results on tasks with safety requirements (G.2), an investigation of the generalisation capabilities of DeepLTL to longer sequences (G.3), two ablation studies on the impact of curriclum learning (G.4) and our model architecture (G.5), investigations on the performance impact of the number of atomic propositions (G.6) and specification complexity (G.7), and a comparison to *RAD-embeddings* (Yalcinkaya et al., 2024), a recent automata-conditioned RL method (G.8). Finally, we show trajectories of trained policies in the *ZoneEnv* and *FlatWorld* environments in Appendix G.9.



(a) G F blue∧G F green (b) F G blue

		GCRL-LTL	DeepLTL
LetterWorld	ψ_1	$6.74_{\pm 1.96}$	$\mathbf{17.84}_{\pm 0.67}$
	ψ_2	$1.49_{\pm 0.31}$	$\mathbf{3.05}_{\pm 0.47}$
ZoneEnv	ψ_3	$3.14_{\pm 0.18}$	$\mathbf{4.16}_{\pm 0.61}$
	ψ_4	$1.31_{\pm 0.05}$	$\mathbf{1.71}_{\pm 0.23}$
FlatWorld	ψ_5	$7.20_{\pm 4.08}$	$\mathbf{12.12}_{\pm 0.80}$
	ψ_6	$4.64_{\pm 0.25}$	$\mathbf{6.98}_{\pm 1.00}$

(c)

Figure 6: Results on infinite-horizon tasks. (a), (b) Example trajectories for infinite-horizon specifications. (c) Performance on various recurrence tasks. We report the average number of visits to accepting states over 500 episodes (i.e. completed cycles), with standard deviations over 5 seeds.

6 Related Work

RL with tasks expressed in LTL has received significant attention in the last few years (Sadigh et al., 2014; De Giacomo et al., 2018; Camacho et al., 2019; Kazemi et al., 2022; Li et al., 2024). Our approach builds on previous works that use LDBAs to augment the state space of the MDP (Hasanbeig et al., 2018; Hahn et al., 2019; 2020; Bozkurt et al., 2020; Voloshin et al., 2022; Hasanbeig et al., 2023; Bagatella et al., 2024; Shah et al., 2024). However, these methods are limited to finding policies for a *single*, fixed specification. In contrast, our approach is realised in a multi-task setting and learns a policy that can zero-shot generalise to arbitrary specifications at test time.

Among the works that consider multiple, previously unseen specifications, many approaches decompose a given task into subtasks, which are then individually completed (Araki et al., 2021; León et al., 2021; 2022; Liu et al., 2024). However, as noted by Vaezipoor et al. (2021) this results in *myopic* behaviour and hence potentially suboptimal solutions. In contrast, our approach takes the entire specification into account by reasoning over temporally extended reach-avoid sequences. Kuo et al. (2020) instead propose to compose RNNs in a way that mirrors formula structure, which however requires learning a non-stationary policy. This is addressed by LTL2Action (Vaezipoor et al., 2021), which encodes the syntax tree of a target specification using a GNN and uses *LTL progression* (Bacchus & Kabanza, 2000) to make the problem Markovian. A similar approach is proposed by Yalcinkaya et al. (2024), who use a GNN to encode tasks represented by deterministic finite automata. We instead extract reach-avoid sequences from Büchi automata, which explicitly lists the possible ways of satisfying a given specification. Furthermore, due to its reliance on LTL progression, LTL2Action is restricted to the finite-horizon fragment of LTL, whereas our approach is able to handle infinite-horizon tasks.

The only previous method we are aware of that can deal with infinite-horizon specifications is GCRL-LTL (Qiu et al., 2023). However, similar to other approaches, GCRL-LTL relies on composing policies for sub-tasks and therefore produces suboptimal behaviour. Furthermore, the approach only considers safety constraints during task execution and not during high-level planning. Recently, Xu & Fekri (2024) proposed *future dependent options* for satisfying arbitrary LTL tasks, which are option policies that depend on future goals. Their method is only applicable to a fragment of LTL that does not support conjunction nor infinite-horizon specifications, and does not consider safety constraints during planning. See Appendix D for an extended discussion of related work.

7 Conclusion

We have introduced *DeepLTL*, a novel approach to the problem of learning policies that can zero-shot execute arbitrary LTL specifications. Our method represents a given specification as a set of reach-avoid sequences of truth assignments, and exploits a general sequence-conditioned policy to execute arbitrary LTL instructions at test time. In contrast to existing techniques, our method can handle infinite-horizon specifications, is non-myopic, and naturally considers safety constraints. Through extensive experiments, we have demonstrated the effectiveness of our approach in practice. This work contributes to the field of LTL-conditioned RL, which has the potential to pave the way towards general AI systems capable of executing arbitrary, well-defined tasks.

ACKNOWLEDGEMENTS

We are grateful for the valuable feedback from the anonymous reviewers. This work was supported by the UKRI AI Hub (EP/Y028872/1) and the ARIA TA1 project SAINT. MJ is funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1).

REFERENCES

Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 424–439, 2014. ISBN 978-3-642-54862-8. doi: 10.1007/978-3-642-54862-8_37.

Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. Policy Synthesis and Reinforcement Learning for Discounted LTL. In Constantin Enea and Akash Lal (eds.), *Computer Aided Verification*, pp. 415–435, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-37706-8. doi: 10.1007/978-3-031-37706-8_21.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 166–175. PMLR, July 2017.

Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan Decastro, Micah Fry, and Daniela Rus. The Logical Options Framework. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 307–317, July 2021.

Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1):123–191, 2000. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00071-5.

Marco Bagatella, Andreas Krause, and Georg Martius. Directed Exploration in Reinforcement Learning from Linear Temporal Logic. In *arXiv*. arXiv, 2024. doi: 10.48550/arXiv.2408.09495.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

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)*, pp. 10349–10355, May 2020. doi: 10.1109/ICRA40945.2020.9196796.

Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Learning Optimal Strategies for Temporal Tasks in Stochastic Games. *IEEE Transactions on Automatic Control*, 69 (11):7387–7402, November 2024. ISSN 1558-2523. doi: 10.1109/TAC.2024.3390848.

Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov Decision Processes Using Learning Algorithms. In Franck Cassez and Jean-François Raskin (eds.), *Automated Technology for Verification and Analysis*, pp. 98–114, 2014. ISBN 978-3-319-11936-6. doi: 10.1007/978-3-319-11936-6_8.

J. R. Büchi. Symposium on Decision Problems: On a Decision Method in Restricted Second Order Arithmetic. In *Studies in Logic and the Foundations of Mathematics*, volume 44 of *Logic, Methodology and Philosophy of Science*, pp. 1–11. Elsevier, January 1966. doi: 10.1016/S0049-237X(09)70564-6.

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*, 6(4):7973–7980, October 2021. ISSN 2377-3766. doi: 10.1109/LRA.2021.3101544.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 6065–6073. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/840.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-4012.

Joëlle Cohen-Chesnot. On the expressive power of temporal logic for infinite words. *Theoretical Computer Science*, 83(2):301–312, June 1991. ISSN 0304-3975. doi: 10.1016/0304-3975(91)90281-6.

Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pp. 854–860, Beijing, China, August 2013. AAAI Press. ISBN 978-1-57735-633-2.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Reinforcement learning for ltlf/ldlf goals. In *arXiv*, 2018.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29:128–136, 2019. ISSN 2334-0843. doi: 10.1609/icaps.v29i1.3549.

Jie Fu and Ufuk Topcu. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*, July 2014. ISBN 978-0-9923747-0-9. doi: 10.15607/RSS.2014.X.039.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pp. 2385–2391. AAAI Press, 2019. ISBN 978-0-9992411-4-1.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-Regular Objectives in Model-Free Reinforcement Learning. In Tomáš Vojnar and Lijun Zhang (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 395–412, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17462-0. doi: 10.1007/978-3-030-17462-0_27.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives. In Dang Van Hung and Oleg Sokolsky (eds.), *Automated Technology for Verification and Analysis*, pp. 108–124, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59152-6. doi: 10.1007/978-3-030-59152-6_6.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Mungojerrie: Linear-Time Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pp. 527–545, 2023. ISBN 978-3-031-30823-9. doi: 10.1007/978-3-031-30823-9_27.

Hosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Certified reinforcement learning with logic guidance. *Artificial Intelligence*, 322:103949, 2023. ISSN 0004-3702. doi: 10.1016/j.artint.2023.103949.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-Constrained Reinforcement Learning. *arXiv*, 2018. doi: 10.48550/arXiv.1801.08099.

Jiaming Ji, Jiayi Zhou, Borong Zhang, Juntao Dai, Xuehai Pan, Ruiyang Sun, Weidong Huang, Yiran Geng, Mickel Liu, and Yaodong Yang. OmniSafe: An Infrastructure for Accelerating Safe Reinforcement Learning Research, May 2023.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A Composable Specification Language for Reinforcement Learning Tasks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional Reinforcement Learning from Logical Specifications. In *Advances in Neural Information Processing Systems*, volume 34, pp. 10026–10039, 2021.

Milad Kazemi and Sadegh Soudjani. Formal Policy Synthesis for Continuous-State Systems via Reinforcement Learning. In *Integrated Formal Methods: 16th International Conference*, pp. 3–21, November 2020. ISBN 978-3-030-63460-5. doi: 10.1007/978-3-030-63461-2_1.

Milad Kazemi, Mateo Perez, Fabio Somenzi, Sadegh Soudjani, Ashutosh Trivedi, and Alvaro Velasquez. Translating Omega-Regular Specifications to Average Objectives for Model-Free Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, pp. 732–741, Richland, SC, May 2022. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-9213-6.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations*, 2015. doi: 10.48550/arXiv.1412.6980.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5604–5610, October 2020. doi: 10.1109/IROS45743.2020.9341325.

Xuan-Bach Le, Dominik Wagner, Leon Witzman, Alexander Rabinovich, and Luke Ong. Reinforcement Learning with LTL and ω-Regular Objectives via Optimality-Preserving Translation to Average Rewards. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning. In *arXiv*. arXiv, 2021. doi: 10.48550/arXiv.2006.08767.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. In a Nutshell, the Human Asked for This: Latent Goals for Following Temporal Specifications. In *International Conference on Learning Representations*, 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 *arXiv*. arXiv, 2024. doi: 10.48550/arXiv.2406.00120.

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments. In *Proceedings of The 7th Conference on Robot Learning*, pp. 1084–1110. PMLR, 2023.

Jason Xinyu Liu, Ankit Shah, Eric Rosen, Mingxi Jia, George Konidaris, and Stefanie Tellex. Skill Transfer for Temporal Task Specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2535–2541, 2024. doi: 10.1109/ICRA57147.2024.10611432.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 5502–5511, July 2022. ISBN 978-1-956792-00-3. doi: 10.24963/ijcai.2022/770.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 6309–6317. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/880.

Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pp. 377–410, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 978-0-89791-404-8. doi: 10.1145/93385.93442.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020. ISSN 1533-7928.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2661–2670. PMLR, July 2017.

Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-Efficient Learning of Natural Language to Linear Temporal Logic Translators for Robot Task Specification. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11554–11561, 2023. doi: 10.1109/ICRA48891.2023.10161125.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pp. 46–57, October 1977. doi: 10.1109/SFCS.1977.32.

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. In *Thirty-Seventh Conference on Neural Information Processing Systems*, November 2023.

Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, and Sanjit A. Seshia. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pp. 1091–1096, December 2014. doi: 10.1109/CDC.2014.7039527.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv*, (preprint), August 2017. doi: 10.48550/arXiv.1707.06347.

Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A. Seshia. LTL-Constrained Policy Optimization with Cycle Experience Replay. In *arXiv*, 2024. doi: 10.48550/arXiv.2404.11578.

Daqian Shao and Marta Kwiatkowska. Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 4180–4189, 2023. doi: 10.24963/IJCAI.2023/465.

John J. Shynk. *Probability, Random Variables, and Random Processes: Theory and Signal Processing Applications*. John Wiley & Sons, 2012. ISBN 978-1-118-39395-6.

Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *Computer Aided Verification*, Lecture Notes in Computer Science, pp. 312–332, 2016. ISBN 978-3-319-41540-6. doi: 10.1007/978-3-319-41540-6_17.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Teaching Multiple Tasks to an RL Agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 452–461, July 2018a.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 2107–2116. PMLR, July 2018b.

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*, 73:173–208, January 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo A. Toro Icarte, and Sheila A. Mcilraith. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10497–10508. PMLR, July 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 *Advances in Neural Information Processing Systems*, volume 30, 2017.

Cameron Voloshin, Hoang Le, Swarat Chaudhuri, and Yisong Yue. Policy Optimization with Linear Temporal Logic Constraints. *Advances in Neural Information Processing Systems*, 35: 17690–17702, December 2022.

Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual Discounting Temporal Logic Counterfactual Experience Replay. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 35137–35150. PMLR, July 2023.

Christopher J C H Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

Duo Xu and Faramarz Fekri. Generalization of temporal logic tasks via future dependent options. *Machine Learning*, August 2024. ISSN 1573-0565. doi: 10.1007/s10994-024-06614-y.

Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit Seshia. Automata Conditioned Reinforcement Learning with Experience Replay. In *NeurIPS 2023 Workshop on Goal-Conditioned Reinforcement Learning*, 2023.

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

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

A LTL SATISFACTION SEMANTICS

The satisfaction semantics of LTL are defined in terms of infinite sequences of truth assignments $a \in 2^{AP}$ (a.k.a. ω-words over 2^{AP}). The satisfaction relation $w \models \varphi$ specifies that ω-word w *satisfies* the specification φ. It is recursively defined as follows (Baier & Katoen, 2008):

$$
\begin{aligned}
&w \models \mathsf{true} && \\
&w \models \mathsf{a} && \text{iff } \mathsf{a} \in w_0 \\
&w \models \varphi \wedge \psi && \text{iff } w \models \varphi \text{ and } w \models \psi \\
&w \models \neg\varphi && \text{iff } w \not\models \varphi \\
&w \models \mathsf{X}\,\varphi && \text{iff } w[1\ldots] \models \varphi \\
&w \models \varphi \,\mathsf{U}\, \psi && \text{iff } \exists j \geq 0 \text{ s.t. } w[j\ldots] \models \psi \text{ and } \forall 0 \leq i < j.\ w[i\ldots] \models \varphi.
\end{aligned}
$$

As noted in the main paper, we can equivalently define the satisfaction semantics via (limit-deterministic) Büchi automata. Formally, for any LTL specification φ we can construct a Büchi automaton that accepts exactly the set $\text{Words}(\varphi) = \{w \in (2^{AP})^\omega \mid w \models \varphi\}$.

B LEARNING PROBABILITY-OPTIMAL POLICIES

B.1 EVENTUAL DISCOUNTING

The technique of *eventual discounting* (Voloshin et al., 2023) can be used to find an optimal policy for a given (single) LTL specification φ in terms of satisfaction probability via reinforcement learning. It optimises the following objective:

$$
\pi^*_{RL} = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \Gamma_t \mathbb{1}[q_t \in \mathcal{F}_{\mathcal{B}_\varphi}]\right], \quad \Gamma_t = \gamma^{c_t}, \quad c_t = \sum_{k=0}^{t} \mathbb{1}[q_k \in \mathcal{F}_{\mathcal{B}_\varphi}],
$$

where c_t counts how often accepting states have been visited up to time step t.

To see why eventual discounting is necessary, consider the *product* MDP $\mathcal{M}^\varphi$ depicted in Figure 7, and adapted from Voloshin et al. (2023). The policy starts in state s_0 and can choose either action a or action b. Action a always leads to an infinite cycle containing an accepting state, and is thus optimal. Action b on the other hand also leads to an infinite cycle with probability 0.99, but may lead to a sink state with probability 0.01.

Let π_a be the (deterministic) policy that chooses a and π_b be the policy that chooses b. Without eventual discounting, we have:

$$
J(\pi_a) = \frac{1}{1-\gamma^2} \qquad \text{and} \qquad J(\pi_b) = \frac{0.99}{1-\gamma},
$$

and thus $J(\pi_b) > J(\pi_a)$ for all $\gamma \in (0.01, 1)$. Hence, maximising (standard) expected discounted return produces a suboptimal policy in terms of satisfaction probability.

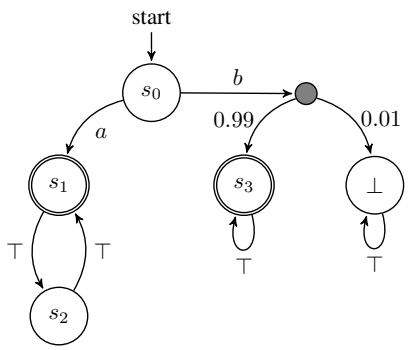


Figure 7: Example product MDP.

Eventual discounting addresses this problem by only discounting visits to accepting states, and not the steps in between. In the previous example, this means that $J(\pi_a) > J(\pi_b)$, in line with the satisfaction probability. See Voloshin et al. (2023) for a formal derivation of a bound on the performance of the return-optimal policy under eventual discounting. We next extend this result to the setting of a distribution ξ over LTL specifications φ.

B.2 Proof of Theorem 1

Given a probability distribution ξ over LTL specifications φ, we employ the standard formulation of goal-conditioned RL to obtain the multi-task eventual discounting objective:

$$\pi^*_{RL} = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\varphi\sim\xi,\\ \tau\sim\pi|\varphi}} \left[\sum_{t=0}^{\infty} \Gamma_t \mathbb{1}[q_t \in \mathcal{F}_{\mathcal{B}_\varphi}]\right], \quad \Gamma_t = \gamma^{c_t}, \quad c_t = \sum_{k=0}^{t} \mathbb{1}[q_k \in \mathcal{F}_{\mathcal{B}_\varphi}]. \tag{5}$$

Theorem 1 provides a bound on the performance of the policy π^*_{RL}. Our proof closely follows the structure of the proof of (Voloshin et al., 2023, Theorem 4.2). We begin with the following Lemma:

Lemma 1. *For any π, $\gamma \in (0,1)$, and $\varphi \in \mathrm{supp}(\xi)$, we have*

$$|(1-\gamma)V^\pi - \Pr(\pi \models \varphi)| \leq \log(\frac{1}{\gamma})O_\pi,$$

where $O_\pi = \mathbb{E}_{\varphi\sim\xi,\tau\sim\pi|\varphi}\left[|\{q \in \tau_q : q \in \mathcal{F}_{\mathcal{B}_\varphi}\}| \,\middle|\, \tau \not\models \varphi\right]$ is the expected number of visits to accepting states for trajectories that do not satisfy a specification.

Proof. The proof follows exactly along the lines of the proof of (Voloshin et al., 2023, Lemma 4.1) with our modified definition of O_π, which includes the expectation over φ. □

We are now ready to prove the main result:

Theorem 1. *For any $\gamma \in (0,1)$ we have*

$$\sup_\pi \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi \models \varphi)] - \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi^*_{RL} \models \varphi)] \leq 2\log(\frac{1}{\gamma}) \sup_\pi O_\pi.$$

Proof. Let $(\pi_i)_{i\in\mathbb{N}}$ be a sequence of policies such that

$$\mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi_i \models \varphi)] \xrightarrow{i\to\infty} \sup_\pi \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi \models \varphi)].$$

By the linearity of expectation, we have

$$\begin{aligned}&\mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi_i \models \varphi)] - \mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi^*_{RL} \models \varphi)]\\ =&\mathop{\mathbb{E}}_{\varphi\sim\xi}[\Pr(\pi_i \models \varphi) - \Pr(\pi^*_{RL} \models \varphi)].\end{aligned}$$

We add and subtract the terms $(1-\gamma)V^{\pi_i}$ and $(1-\gamma)V^{\pi^*_{RL}}$ and apply the triangle inequality to obtain

$$\begin{aligned}\leq \left|\mathop{\mathbb{E}}_{\varphi}[\Pr(\pi_i \models \varphi) - (1-\gamma)V^{\pi_i}]\right| + \left|\mathop{\mathbb{E}}_{\varphi}\left[\Pr(\pi^*_{RL} \models \varphi) - (1-\gamma)V^{\pi^*_{RL}}\right]\right|\\ + \mathop{\mathbb{E}}_{\varphi}\left[(1-\gamma)V^{\pi_i} - (1-\gamma)V^{\pi^*_{RL}}\right].\end{aligned} \tag{6}$$

The last term is negative since

$$\begin{aligned}\mathop{\mathbb{E}}_{\varphi}\left[(1-\gamma)V^{\pi_i} - (1-\gamma)V^{\pi^*_{RL}}\right] &= (1-\gamma)\mathop{\mathbb{E}}_{\varphi}\left[V^{\pi_i} - V^{\pi^*_{RL}}\right]\\ &= (1-\gamma)(V^{\pi_i} - V^{\pi^*_{RL}})\\ &\leq 0 \qquad \text{(by the definition of } \pi^*_{RL}).\end{aligned}$$

Note that for any random variable X, again by the triangle inequality,

$$|\,\mathbb{E}[X]| = \left| \sum_x x \Pr(X = x) \right| \leq \sum_x |x| \Pr(X = x) = \mathbb{E}[|X|],$$

and we can hence continue from Equation 6 by applying Lemma 1 as follows (where we utilise the fact the expectations respect inequalities):

$$\begin{aligned}
(6) &\leq \mathop{\mathbb{E}}_{\varphi} \left[\log(\frac{1}{\gamma})(O_{\pi_i} + O_{\pi^*_{RL}}) \right] \\
&\leq \mathop{\mathbb{E}}_{\varphi} \left[2 \log(\frac{1}{\gamma}) \sup_{\pi} O_{\pi} \right] \\
&= 2 \log(\frac{1}{\gamma}) \sup_{\pi} O_{\pi},
\end{aligned}$$

which, together with taking the limit as $i \to \infty$, concludes the proof. □

B.3 DeepLTL with eventual discounting

DeepLTL can be readily extended with eventual discounting for settings in which satisfaction probability is the primary concern, and efficiency is less important. In this case, we want to use our approach to approximate a solution to Equation 5.

To do so, we only need to assume access to the distribution ξ over LTL formulae. During training, we sample specifications $\varphi \sim \xi$ and train the sequence-conditioned policy using all reach-avoid sequences extracted from $\mathcal{B}_\varphi$. Crucially, we extend each step of a reach-avoid sequence to include an additional Boolean flag that specifies whether the corresponding LDBA transition leads to an accepting state. These flags are given as input to the policy network, and are used to compute the eventual discounting objective. The rest of our approach remains unchanged. We leave an experimental investigation of this scheme for future work.

C Discounted LTL

Discounted LTL (Almagor et al., 2014) extends LTL with discounting semantics. This allows one to specify not only that a system should satisfy a given specification (as in classical LTL), but also *how well* it should do so. The main idea is to replace the "until" operator with a discounted version that assigns a quantitative satisfaction value in $[0, 1]$ to a system trace. Intuitively, the longer the system takes to satisfy a requirement, the smaller the assigned satisfaction value.

The problem of finding an optimal policy with respect to a discounted LTL formula in an unknown MDP has been thoroughly investigated by Alur et al. (2023), and is closely related to our problem of efficient LTL satisfaction (Problem 1). Indeed, for simple formulae such as $\varphi = \mathsf{F}_\gamma\, a$ (where F_γ denotes the γ-discounted version of the F operator) it is easy to see that Equation 2 corresponds exactly to maximising the quantitative satisfaction value of φ. We leave a fuller investigation of the relationship between discounting rewards from accepting LDBA states and the semantics of discounted LTL, as well as an integration of our approach with the reward scheme proposed by Alur et al. (2023), for future work.

D Extended related work

The field of RL with LTL specifications has attracted significant attention in the last few years. Here we provide a more detailed overview of work in this domain and discuss how it relates to our approach.

RL with a single LTL specification. Early works on RL with LTL specifications relied on estimating a model of the underlying MDP, and then solving this model for a probability-maximising policy (Fu & Topcu, 2014; Brázdil et al., 2014; Sadigh et al., 2014). A model-free approach based on Q-learning (Watkins, 1989; Watkins & Dayan, 1992) was introduced by Hasanbeig et al. (2018), who

proposed to use LDBAs to keep track of formula satisfaction. Subsequent works extend this approach, providing stronger convergence guarantees (Hahn et al., 2019; Bozkurt et al., 2020; Hahn et al., 2023; Voloshin et al., 2023), improved sample efficiency (Hahn et al., 2020; Kazemi & Soudjani, 2020; Cai et al., 2021; Shao & Kwiatkowska, 2023; Shah et al., 2024; Bagatella et al., 2024), or guarantees in adversarial environments (Bozkurt et al., 2024). Alternative methods reduce the problem to an RL objective with limit-average rewards instead of the standard discounted setting (Kazemi et al., 2022; Le et al., 2024). A variety of works also consider LTL_f (De Giacomo & Vardi, 2013) or similar specification languages over *finite* traces, such as reward machines (Toro Icarte et al., 2018b; De Giacomo et al., 2018; 2019; Jothimurugan et al., 2019; 2021; Toro Icarte et al., 2022).

These approaches all consider only a *single*, fixed LTL specification, i.e. they learn a policy that maximises the probability of satisfying a given formula φ. In contrast, our approach is realised in a multi-task RL setting: we focus on learning a task-conditional policy that can zero-shot execute arbitrary LTL specifications at test time.

Generalising to multiple tasks. Toro Icarte et al. (2018a) were among the first to consider the problem of training a policy to complete multiple different tasks expressed in LTL. They propose a hierarchical algorithm based on Q-learning, which composes policies trained on subtasks. However, their approach is not able to generalise to novel formulae at test time, since these might consist of subtasks that the agent has not seen during training. Kuo et al. (2020) instead propose to leverage goal-conditioned RL with a compositional RNN architecture that consists of one RNN for every element in the syntax tree of a given LTL formula. While this method is shown to be able to generalise to tasks outside of the training distribution, it requires learning a non-stationary policy, which is known to be challenging (Vaezipoor et al., 2021).

León et al. (2021) introduce a different method for tasks expressed in a sub-fragment of LTL_f. They first employ a reasoning module to extract propositions that make progress towards solving the given task. These propositions are then achieved by a trained goal-conditioned policy. Similarly, Liu et al. (2024) propose a transfer algorithm that first trains a number of options on a set of training instructions, and then composes them at test time to achieve novel tasks. However, as noted by Vaezipoor et al. (2021) these approaches are inherently *myopic*: they do not take future propositions into account when executing the next subtask, and hence can produce suboptimal solutions. Instead, Vaezipoor et al. (2021) propose to directly encode the syntax tree of a given LTL formula using a GNN and predict actions based on the learned representations. To deal with the non-Markovian nature of LTL, they employ *LTL progression* (Bacchus & Kabanza, 2000). A more direct modification to the work of León et al. (2021) is proposed by Xu & Fekri (2024), who train *future-dependent* options for every proposition. These option policies are conditioned not only on the proposition to be achieved next, but also on the remaining propositions that need to be satisfied in the future. Qiu et al. (2023) introduce the first approach that can handle ω-regular specifications. Their technique is based on training goal-conditioned policies $\pi(\cdot|s, p)$ to achieve arbitrary propositions $p \in AP$ in the environment. At test time, they employ a planning procedure to select a sequence of propositions to satisfy and finally execute the according low-level policies.

Our approach differs in a variety of ways from these previous methods: we leverage the structure of Büchi automata to find possible ways of satisfying a given specification. By operating on Büchi automata, our method can naturally handle ω-regular specifications, which only GCRL-LTL is also capable of. However, in comparison to GCRL-LTL our method is non-myopic, since it incorporates the temporally extended structure of tasks via sequences of reach-avoid assignments. Additionally, compared to other methods that employ a high-level planning procedure (Qiu et al., 2023; Xu & Fekri, 2024) our approach considers safety requirements when selecting the optimal reach-avoid sequence, yielding plans that are more likely to be able to be executed without safety violations by the policy.

Automata-conditioned RL. A recent line of work related to LTL objectives is *automata-conditioned RL* (Yalcinkaya et al., 2023; 2024). Instead of using LTL to specify tasks, this line of research directly employs (compositions of) deterministic finite automata (DFAs). Yalcinkaya et al. (2024) explore this setting in a multi-task RL context, and propose to train a goal-conditioned policy conditioned on learned embeddings of DFAs. They also introduce the class of *reach-avoid derived* (RAD) DFAs, which they use for pre-training the automata embeddings.

Compared to our approach, automata-conditioned RL is strictly less expressive, since every DFA can trivially be represented as an LDBA (including objectives not expressible by LTL (Cohen-Chesnot, 1991)), but the converse is not true. In particular, DFAs cannot capture infinite-horizon tasks. Additionally, in contrast to RAD-embeddings (Yalcinkaya et al., 2024), we do not compute an embedding for the entire automaton, but instead extract satisfying reach-avoid sequences and select the optimal one according to the learned value function. This separates high-level reasoning (how to satisfy the specification) from low-level reasoning (how to act in the MDP) and allows the goal-conditioned policy to focus on achieving one particular sequence of propositions. We experimentally compare our approach to RAD-embeddings in Appendix G.8.

E COMPUTING ACCEPTING CYCLES

The algorithm to compute paths to accepting cycles is listed in Algorithm 1. The algorithm is based on depth-first search and keeps track of the currently visited path in order to extract all possible paths to accepting cycles.

Algorithm 1 Computing paths to accepting cycles

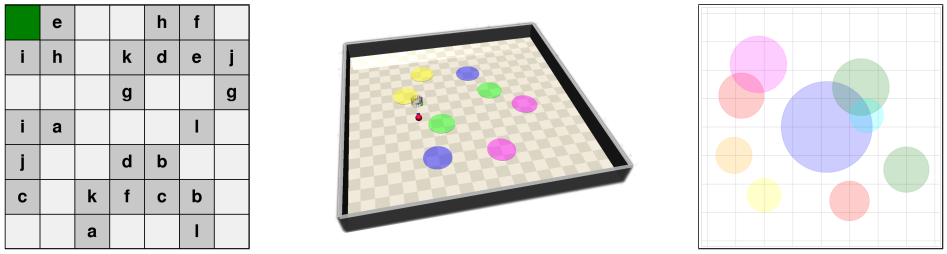
Require:
An LDBA $B = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E})$ and current state q.
procedure DFS(q, p, i) ▷ i is in the index of the last seen accepting state, or -1 otherwise
  $P \leftarrow \emptyset$
  if $q \in \mathcal{F}$ **then**
    $i \leftarrow |p|$
  end if
  for all $a \in 2^{AP} \cup \{\varepsilon\}$ **do**
    $p' \leftarrow [p, q]$
    $q' \leftarrow \delta(q, a)$
    if $q' \in p$ **then**
      if index of q' in $p \leq i$ **then**
        $P = P \cup \{p'\}$
      end if
    else
      $P = P \cup$ DFS(q', p', i)
    end if
  end for
  return P
end procedure
$i \leftarrow 0$ if $q \in \mathcal{F}$ else $i \leftarrow -1$
return DFS(q, [], i)

F EXPERIMENTAL DETAILS

F.1 ENVIRONMENTS

LetterWorld. The *LetterWorld* environment has been introduced by Vaezipoor et al. (2021). It is a 7×7 grid world that contains 12 randomly placed letters corresponding to atomic propositions. Each letter appears twice, i.e. 24 out of the 49 squares are occupied, and there are thus multiple ways of solving any given task. The agent observes the full grid from an egocentric view. At each step, the agent can move up, right, down, or left. If it moves out of bounds, the agent is immediately placed on the opposite end of the grid. See Figure 8a for an illustration of the *LetterWorld* environment.

ZoneEnv. We adapt the *ZoneEnv* environment introduced by Vaezipoor et al. (2021). The environment is a walled plane with 8 circular regions ("zones") that have four different colours and form the atomic propositions. Our implementation is based on the Safety Gymnasium suite (Ji et al., 2023) and uses the *Point* robot, which has a continuous action space for acceleration and steering. The environment features a high-dimensional state space based on lidar information about the zones, and



(a) LetterWorld (b) ZoneEnv (c) FlatWorld

Figure 8: Visualisations of environments.

data from other sensors. Both the zone and robot positions are randomly sampled at the beginning of each episode. If the agent at any point touches a wall, it receives a penalty and the episode is immediately terminated. A visualisation of the *ZoneEnv* environment is provided in Figure 8b.

FlatWorld. The *FlatWorld* environment (Voloshin et al., 2023; Shah et al., 2024) consists of a two-dimensional continuous world ($\mathcal{S} = [-2, 2]^2$) with a discrete action space. Atomic propositions are given by various coloured regions. Importantly, these regions overlap in various places, which means that multiple propositions can hold true at the same time. The initial agent position is sampled randomly from the space in which no propositions are true. At each time step, the agent can move in one of the 8 compass directions. If it leaves the boundary of the world, the agent receives a penalty and the episode is terminated prematurely. Figure 8c shows a visualisation of the *FlatWorld* environment.

F.2 Testing specifications

Tables 2 and 3 list the finite and infinite-horizon specifications used in our evaluation, respectively.

F.3 Hyperparameters

Neural network architectures. Our choice of neural network architectures is similar to previous work (Vaezipoor et al., 2021). For DeepLTL and LTL2Action, we employ a fully connected actor network with [64, 64, 64] units and ReLU as the activation function. The critic has network structure [64, 64] and uses Tanh activations in *LetterWorld* and *ZoneEnv*, and ReLU activations in *FlatWorld*. The actor is composed with a softmax layer in discrete action spaces, and outputs the mean and standard deviation of a Gaussian distribution in continuous action spaces. GCRL-LTL uses somewhat larger actor and critic networks with structure [512, 1024, 256] and ReLU activations in the *ZoneEnv* environment.

The observation module is environment-specific. For the *ZoneEnv* and *FlatWorld* environments, it consists of a simple fully connected network with [128, 64] units and Tanh activations, or [16, 16] units and ReLU activations, respectively. GCRL-LTL instead uses a simple projection of the input to dimensionality 100 in *ZoneEnv*. For *LetterWorld*, the observation module is a CNN with 16, 32, and 64 channels in three hidden layers, a kernel size of 2×2, stride of 1, no padding, and ReLU activations.

Finally, the sequence module consists of learned embeddings ϕ of dimensionality 32 in *LetterWorld*, and 16 in *ZoneEnv* and *FlatWorld*. The non-linear transformation ρ is a fully connected network with [32, 32] units in *LetterWorld*, and [32, 16] units in *ZoneEnv* and *FlatWorld*. We use ReLU activations throughout for the sequence module. For the baselines, we use the hyperparameters reported in the respective papers (Vaezipoor et al., 2021; Qiu et al., 2023).

PPO hyperparameters. The hyperparameters for PPO (Schulman et al., 2017) are listed in Table 4. We use the Adam optimiser (Kingma & Ba, 2015) for all methods and environments.

Additional hyperparameters. LTL2Action requires an LTL task sampler to sample a random training specification at the beginning of each episode. We follow Vaezipoor et al. (2021) and

Table 2: *Complex* finite-horizon specifications used in our evaluation.

LetterWorld	φ_1	$\mathsf{F}\,(\mathsf{a} \land (\neg \mathsf{b}\ \mathsf{U}\ \mathsf{c})) \land \mathsf{F}\,\mathsf{d}$
	φ_2	$\mathsf{F}\,\mathsf{d} \land (\neg \mathsf{f}\ \mathsf{U}\ (\mathsf{d} \land \mathsf{F}\,\mathsf{b}))$
	φ_3	$\mathsf{F}\,((\mathsf{a} \lor \mathsf{c} \lor \mathsf{j}) \land \mathsf{F}\,\mathsf{b}) \land \mathsf{F}\,(\mathsf{c} \land \mathsf{F}\,\mathsf{d}) \land \mathsf{F}\,\mathsf{k}$
	φ_4	$\neg \mathsf{a}\ \mathsf{U}\ (\mathsf{b} \land (\neg \mathsf{c}\ \mathsf{U}\ (\mathsf{d} \land (\neg \mathsf{e}\ \mathsf{U}\ \mathsf{f}))))$
	φ_5	$((\mathsf{a} \lor \mathsf{b} \lor \mathsf{c} \lor \mathsf{d}) \Rightarrow \mathsf{F}\,(\mathsf{e} \land (\mathsf{F}\,(\mathsf{f} \land \mathsf{F}\,\mathsf{g})))) \ \mathsf{U}\ (\mathsf{h} \land \mathsf{F}\,\mathsf{i})$
ZoneEnv	φ_6	$\mathsf{F}\,(\mathsf{green} \land (\neg \mathsf{blue}\ \mathsf{U}\ \mathsf{yellow})) \land \mathsf{F}\,\mathsf{magenta}$
	φ_7	$\mathsf{F}\,\mathsf{blue} \land (\neg \mathsf{blue}\ \mathsf{U}\ (\mathsf{green} \land \mathsf{F}\,\mathsf{yellow}))$
	φ_8	$\mathsf{F}\,(\mathsf{blue} \lor \mathsf{green}) \land \mathsf{F}\,\mathsf{yellow} \land \mathsf{F}\,\mathsf{magenta}$
	φ_9	$\neg(\mathsf{magenta} \lor \mathsf{yellow})\ \mathsf{U}\ (\mathsf{blue} \land \mathsf{F}\,\mathsf{green})$
	φ_{10}	$\neg \mathsf{green}\ \mathsf{U}\ ((\mathsf{blue} \lor \mathsf{magenta}) \land (\neg \mathsf{green}\ \mathsf{U}\ \mathsf{yellow}))$
	φ_{11}	$((\mathsf{green} \lor \mathsf{blue}) \Rightarrow (\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{magenta}))\ \mathsf{U}\ \mathsf{yellow}$
FlatWorld	φ_{12}	$\mathsf{F}\,((\mathsf{red} \land \mathsf{magenta}) \land \mathsf{F}\,((\mathsf{blue} \land \mathsf{green}) \land \mathsf{F}\,\mathsf{yellow}))$
	φ_{13}	$\mathsf{F}\,(\mathsf{orange} \land (\neg \mathsf{red}\ \mathsf{U}\ \mathsf{magenta}))$
	φ_{14}	$(\neg \mathsf{red}\ \mathsf{U}\ (\mathsf{green} \land \mathsf{blue} \land \mathsf{aqua})) \land \mathsf{F}\,(\mathsf{orange} \land (\mathsf{F}\,(\mathsf{red} \land \mathsf{magenta})))$
	φ_{15}	$((\neg \mathsf{yellow} \land \neg \mathsf{orange})\ \mathsf{U}\ (\mathsf{green} \land \mathsf{blue})) \land (\neg \mathsf{green}\ \mathsf{U}\ \mathsf{magenta})$
	φ_{16}	$(\mathsf{blue} \Rightarrow \mathsf{F}\,\mathsf{magenta})\ \mathsf{U}\ (\mathsf{yellow} \lor ((\mathsf{green} \land \mathsf{blue}) \land \mathsf{F}\,\mathsf{orange}))$

Table 3: Infinite-horizon specifications used in our evaluation.

LetterWorld	ψ_1	$\mathsf{G}\,\mathsf{F}\,(\mathsf{e} \land (\neg \mathsf{a}\ \mathsf{U}\ \mathsf{f}))$
	ψ_2	$\mathsf{G}\,\mathsf{F}\,\mathsf{a} \land \mathsf{G}\,\mathsf{F}\,\mathsf{b} \land \mathsf{G}\,\mathsf{F}\,\mathsf{c} \land \mathsf{G}\,\mathsf{F}\,\mathsf{d} \land \mathsf{G}\,(\neg \mathsf{e} \land \neg \mathsf{f})$
ZoneEnv	ψ_3	$\mathsf{G}\,\mathsf{F}\,\mathsf{blue} \land \mathsf{G}\,\mathsf{F}\,\mathsf{green}$
	ψ_4	$\mathsf{G}\,\mathsf{F}\,\mathsf{blue} \land \mathsf{G}\,\mathsf{F}\,\mathsf{green} \land \mathsf{G}\,\mathsf{F}\,\mathsf{yellow} \land \mathsf{G}\,\neg \mathsf{magenta}$
FlatWorld	ψ_5	$\mathsf{G}\,\mathsf{F}\,(\mathsf{blue} \land \mathsf{green}) \land \mathsf{G}\,\mathsf{F}\,(\mathsf{red} \land \mathsf{magenta})$
	ψ_6	$\mathsf{G}\,\mathsf{F}\,(\mathsf{aqua} \land \mathsf{blue}) \land \mathsf{G}\,\mathsf{F}\,\mathsf{red} \land \mathsf{G}\,\mathsf{F}\,\mathsf{yellow} \land \mathsf{G}\,\neg \mathsf{green}$

Table 4: Hyperparameters for PPO. Dashes (—) indicate that the hyperparameter value is the same across all three methods.

		LTL2Action	GCRL-LTL	DeepLTL
LetterWorld	Number of processes	—	16	—
	Steps per process per update	—	128	—
	Epochs	—	8	—
	Batch size	—	256	—
	Discount factor	—	0.94	—
	GAE-λ	—	0.95	—
	Entropy coefficient	—	0.01	—
	Value loss coefficient	—	0.5	—
	Max gradient norm	—	0.5	—
	Clipping (ϵ)	—	0.2	—
	Adam learning rate	—	0.0003	—
	Adam epsilon	—	1e-08	—
ZonesEnv	Number of processes	—	16	—
	Steps per process per update	4096	3125	4096
	Epochs	—	10	—
	Batch size	2048	1000	2048
	Discount factor	—	0.998	—
	GAE-λ	—	0.95	—
	Entropy coefficient	—	0.003	—
	Value loss coefficient	—	0.5	—
	Max gradient norm	—	0.5	—
	Clipping (ϵ)	—	0.2	—
	Adam learning rate	—	0.0003	—
	Adam epsilon	—	1e-08	—
FlatWorld	Number of processes	—	16	—
	Steps per process per update	—	4096	—
	Epochs	—	10	—
	Batch size	—	2048	—
	Discount factor	—	0.98	—
	GAE-λ	—	0.95	—
	Entropy coefficient	—	0.003	—
	Value loss coefficient	—	0.5	—
	Max gradient norm	—	0.5	—
	Clipping (ϵ)	—	0.2	—
	Adam learning rate	—	0.0003	—
	Adam epsilon	—	1e-08	—

Table 5: Evaluation results of trained policies on *persistence* tasks. We report the average number of time steps for which the policy successfully remains in the target region after executing the ε-action. Results are averaged over 5 seeds and 500 episodes per seed. "$\pm$" indicates the standard deviation over seeds.

	GCRL-LTL	DeepLTL
$\mathsf{F\,G\,blue}$	$265.53_{\pm 94.54}$	$\mathbf{562.81}_{\pm 136.28}$
$\mathsf{F\,G\,blue} \land \mathsf{F}\,(\mathsf{yellow} \land \mathsf{F\,green})$	$178.12_{\pm 62.88}$	$\mathbf{336.81}_{\pm 069.43}$
$\mathsf{F\,G\,magenta} \land \mathsf{G}\,\neg\mathsf{yellow}$	$406.52_{\pm 75.22}$	$\mathbf{587.98}_{\pm 123.63}$
$\mathsf{G}\,((\mathsf{green} \lor \mathsf{yellow}) \Rightarrow \mathsf{F\,blue}) \land \mathsf{F\,G}\,(\mathsf{green} \lor \mathsf{magenta})$	$380.49_{\pm 84.74}$	$\mathbf{570.37}_{\pm 138.98}$

sample specifications from the space of *reach/avoid* tasks. We also experimented with sampling more complex specifications, but found this detrimental to performance. For GCRL-LTL, we set the value threshold σ to 0.9 in the *ZoneEnv* environment, and 0.92 in *LetterWorld* and *FlatWorld*. The threshold λ for strict negative assignments in DeepLTL is set to 0.4 across experiments.

F.4 Training curricula

We design training curricula in order to gradually expose the policy to more challenging tasks. The general structure of the curricula is the same across environments: we start with simple and short reach-avoid sequences, and move to more complicated sequences once the policy achieves satisfactory performance.

LetterWorld. In the *LetterWorld* environment, the first curriculum stage consists of reach-avoid tasks of depth 1 with single propositions, e.g. $(\{\{a\}\}, \{\{b\}\}\})$. Once the policy achieves an average satisfaction rate of 95% on these sequences, we move to the next curriculum stage, in which the depth is still 1, and $|A_1^+| \leq 2, |A_1^-| \leq 2$. The next stage consists of the same tasks with a length of 2. The final curriculum stage consists of length-3 sequences with $|A_i^+| \leq 2, |A_i^-| \leq 3$.

ZoneEnv. For *ZoneEnv*, the first stages consist of first only reach-sequences of length up to 2 (i.e. $A_i^- = \emptyset$) and then reach-avoid sequences of length up to 2. We then increase the cardinality of the positive and negative assignments, while introducing sequences aligned with reach-stay tasks, e.g.

$$\Big(\Big(\{\{\mathsf{green}\}\}, 2^{AP} \setminus \{\mathsf{green}\}\Big), \ldots\Big).$$

FlatWorld. In *FlatWorld*, we first sample a mixture of reach- and reach-avoid sequences of depth up to 2. Once the policy achieves a success rate of 80%, we increase the cardinality of the positive and negative assignments up to 2.

G Further experimental results

G.1 Results on persistence tasks

Persistence (a.k.a. *reach-stay*) tasks of the form $\mathsf{F\,G\,a}$ specify that a proposition needs to be true forever from some point on. We evaluate our approach on these tasks in the *ZoneEnv* environment, which features a continuous action space and is thus particularly challenging for reach-stay specifications. Table 5 compares the performance of our approach to the relevant baseline GCRL-LTL, where we report the average number of steps for which the agent successfully remains in the target region after executing the ε-action as the performance metric.[3] The results confirm that our method can successfully handle complex persistence tasks, and performs better than the baseline.

[3]When the policy executes the ε-action, this indicates that the target proposition should from then on be true forever (see e.g. q_2 in Figure 2). Since GCRL-LTL does not explicitly model the ε-actions required for $\mathsf{F\,G}$ specifications, we employ an approximation and execute the ε-action upon entering the target region for the first time, i.e. when the agent first enters the blue region for the task $\mathsf{F\,G\,blue}$.

Table 6: Evaluation results of trained policies on long reachability (R-12) and reach-avoid (RA-6) tasks. We report the *success rate* (SR) and average number of steps to satisfy the task (μ). Results are averaged over 5 seeds and 500 episodes per seed. "$\pm$" indicates the standard deviation over seeds.

	LTL2Action		GCRL-LTL		DeepLTL	
	SR (↑)	μ (↓)	SR (↑)	μ (↓)	SR (↑)	μ (↓)
R-12	$0.89_{\pm 0.14}$	$47.80_{\pm 7.74}$	$0.93_{\pm 0.05}$	$60.15_{\pm 2.59}$	$\mathbf{0.98}_{\pm 0.01}$	$\mathbf{43.33}_{\pm 0.45}$
RA-6	$0.13_{\pm 0.03}$	$56.37_{\pm 0.89}$	$0.62_{\pm 0.09}$	$30.64_{\pm 1.59}$	$\mathbf{0.95}_{\pm 0.01}$	$\mathbf{24.94}_{\pm 0.36}$

G.2 Adversarial tasks with safety constraints

We next demonstrate the advantages of our approach in terms of handling safety constraints on an adversarial task. This task is specifically designed to require considering safety requirements during high-level planning. We consider the configuration of the *ZoneEnv* environment depicted in Figure 9 and the specification $\varphi = \neg\mathsf{blue}\ \mathsf{U}\ (\mathsf{green} \vee \mathsf{yellow}) \wedge \mathsf{F}\,\mathsf{magenta}$.

Figure 9 shows a number of sample trajectories generated by DeepLTL (top row) and GCRL-LTL (bottom row). Evidently, GCRL-LTL fails at satisfying the task, since it decides on reaching the green region during high-level planning without considering the safety constraint $\neg\mathsf{blue}$. In contrast, DeepLTL considers this safety requirement when selecting the optimal reach-avoid sequence, and hence chooses to reach the yellow region instead.

Quantitatively, GCRL-LTL only succeeds in satisfying the task in 9.2% of cases, whereas DeepLTL achieves a success rate of 79.6% (averaged over 5 seeds and 100 episodes per seed). While LTL2Action does not have a high-level planning component, it considers the entire task including safety constraints via its GNN encoding, and averages a success rate of 50.2%.

G.3 Generalisation to longer sequences

In this section, we investigate the ability of our approach to generalise to longer sequences. Our evaluation so far has already demonstrated that our method can successfully learn general behaviour for satisfying LTL specifications by only training on simple reach-avoid sequences. We now extend our analysis by specifically investigating tasks that require a large number of steps to solve.

In particular, we consider the following two task spaces in the *LetterWorld* environment: sequential reachability objectives of depth 12 (i.e. $\mathsf{F}\,(p_1 \wedge (\mathsf{F}\,p_2 \wedge \ldots \wedge \mathsf{F}\,p_{12}))$ and reach-avoid specifications of depth 6 (i.e. $\neg p_1\ \mathsf{U}\ (p_2 \wedge \ldots \wedge (\neg p_{11}\ \mathsf{U}\ p_{12})))$. Note that the longest reach-avoid sequences sampled during training are of length 3. In Table 6, we report the results of our method and the baselines on randomly sampled tasks from the task spaces described above. We observe that our approach generalises well to longer sequences and outperforms the baselines in terms of satisfaction probability and efficiency.

G.4 Ablation study: curriculum learning

We conduct an ablation study in the *LetterWorld* environment to investigate the impact of curriculum learning. We train our method without any training curriculum by directly sampling random reach-avoid sequences of length 3, with potentially multiple propositions to reach and avoid at each stage. The evaluation curves on *reach/avoid* specifications over training (Figure 10) demonstrate that curriculum training improves sample efficiency and reduces variance across seeds.

G.5 Ablation study: sequence module

We conduct a further ablation study to investigate the impact of replacing the RNN in the sequence module with a Transformer (Vaswani et al., 2017) model. Since the reach-avoid sequences arising from common LTL tasks are generally relatively short (2-15 tokens), we expect the simplicity and effectiveness of RNNs to outweigh the benefits of the more complicated Transformer architecture in our setting. For our ablation study, we use a Transformer encoder to learn an embedding of the reach-avoid sequence as the final representation of a special [CLS] token, and keep the rest of the

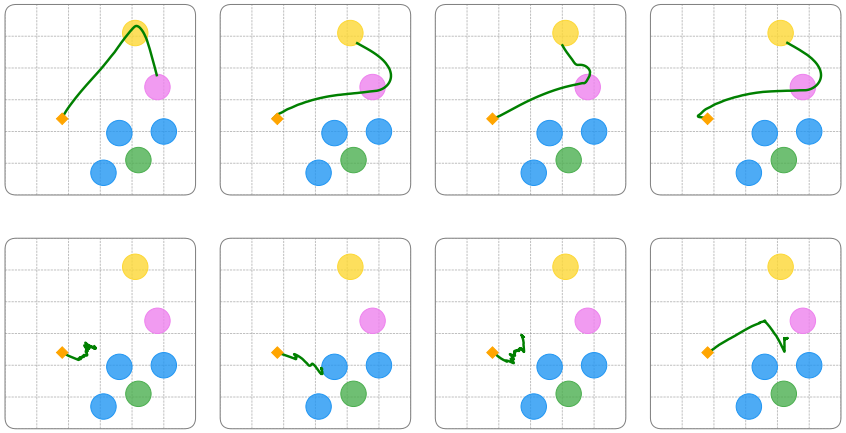


Figure 9: Trajectories of DeepLTL (top row) and GCRL-LTL (bottom row) on the adversarial configuration of the *ZoneEnv* environment. The specification to be completed is $\varphi = \neg\text{blue}\ \mathsf{U}\ (\text{green} \vee \text{yellow}) \wedge \mathsf{F}\ \text{magenta}$. GCRL-LTL ignores safety constraints during high-level planning, and hence performs poorly. In contrast, DeepLTL yields trajectories that satisfy the specification.

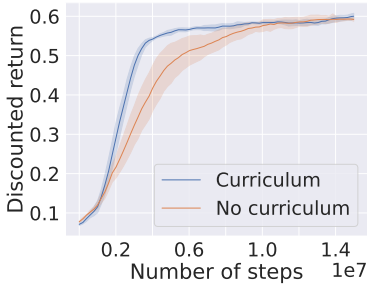


Figure 10: Evaluation curves of training with and without a curriculum on the *LetterWorld* environment. Each datapoint is collected by averaging the discounted return of the policy across 50 episodes with randomly sampled *reach/avoid* specifications, and shaded areas indicate 90% confidence intervals over 5 different random seeds.

Table 7: Comparison of Transformer- and GRU-based sequence modules. We report the *success rate* (SR) and average number of steps to satisfy the task (μ). Results are averaged over 5 seeds and 500 episodes per seed. "$\pm$" indicates the standard deviation over seeds.

	DeepLTL-Transformer		DeepLTL-GRU	
	SR ($\uparrow$)	μ ($\downarrow$)	SR ($\uparrow$)	μ ($\downarrow$)
φ_6	$0.78_{\pm 0.09}$	$462.03_{\pm 52.60}$	$\mathbf{0.92}_{\pm 0.06}$	$\mathbf{303.38}_{\pm 19.43}$
φ_7	$0.46_{\pm 0.04}$	$438.24_{\pm 42.35}$	$\mathbf{0.91}_{\pm 0.03}$	$\mathbf{299.95}_{\pm 09.47}$
φ_8	$0.93_{\pm 0.03}$	$357.34_{\pm 51.86}$	$\mathbf{0.96}_{\pm 0.04}$	$\mathbf{259.75}_{\pm 08.07}$
φ_9	$0.74_{\pm 0.02}$	$275.11_{\pm 31.56}$	$\mathbf{0.90}_{\pm 0.03}$	$\mathbf{203.36}_{\pm 14.97}$
φ_{10}	$0.80_{\pm 0.15}$	$260.58_{\pm 29.48}$	$\mathbf{0.91}_{\pm 0.02}$	$\mathbf{187.13}_{\pm 10.61}$
φ_{11}	$0.90_{\pm 0.07}$	$137.46_{\pm 21.26}$	$\mathbf{0.98}_{\pm 0.01}$	$\mathbf{106.21}_{\pm 07.88}$

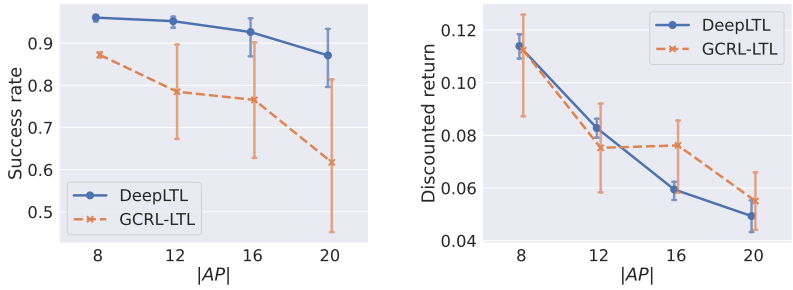


Figure 11: Impact of varying the number of atomic propositions on success rate (left) and discounted return (right). Each datapoint is an average over 500 episodes with randomly sampled *reach/avoid* specifications, and error bars indicate 90% confidence intervals over 5 different random seeds.

model architecture the same. We use an encoder model with an embedding size of 32, 4 attention heads, and a 512-dimensional MLP with dropout of 0.1. Despite this being a very small Transformer model, it still has more than 5 times the number of parameters of the GRU.

We report evaluation results for *complex* specifications in the *ZoneEnv* environment in Table 7, which strongly support our hypothesis that a simpler RNN-based sequence module is more effective.

G.6 Impact of the number of atomic propositions

We conduct an experiment to evaluate the impact of the number of atomic propositions. For this, we consider modified versions of the *FlatWorld* environment with $n_r \in \{8, 12, 16, 20\}$ regions of different colours (that is, in each version we have n_r regions and $|AP| = n_r$). The state space of the *FlatWorld* environment does not depend on the atomic propositions, therefore we can use the same model architecture throughout the experiment.

The results are shown in Figure 11. We observe that policy performance generally decreases as the number of atomic propositions increases, both for DeepLTL and GCRL-LTL. This result is expected, since we increase the number of possible tasks while keeping the number of policy parameters fixed. We also observe that the variance of success becomes larger as the number of atomic propositions increases. This can be explained by the fact that, in the case of a large number of atomic propositions, the goal-conditioned RL process may (for some random seeds) yield a policy that cannot consistently satisfy every single proposition. If there exists a proposition $a \in AP$ that the policy has failed to learn how to satisfy, the policy will consequently not be successful in completing any task that requires a to be achieved at any stage, resulting in a low success rate. Finally, we observe that the performance reduction for DeepLTL generally is less than or similar to the performance reduction for GCRL-LTL.

G.7 Impact of specification complexity

In this section, we provide additional insights regarding the performance of our approach with respect to the complexity of the given specification. An advantage of our method is the representation of tasks as reach-avoid sequences. This representation explicitly lists the possible ways of satisfying a specification, making our approach in principle applicable to arbitrarily complex formulae. As an example, we consider the following specification in the *LetterWorld* environment:

$$(\mathsf{a} \Rightarrow \mathsf{F}\,(\mathsf{b} \wedge (\mathsf{F}\,(\mathsf{c} \wedge \mathsf{F}\,\mathsf{d})))) \;\mathsf{U}\; \big((\mathsf{F}\,((\mathsf{k} \vee \mathsf{b}) \wedge ((\neg\mathsf{b} \wedge \neg\mathsf{d} \wedge \neg\mathsf{j}) \;\mathsf{U}\; \mathsf{l}))) \wedge (\neg\mathsf{h} \;\mathsf{U}\; \mathsf{k})\big) \vee$$
$$\mathsf{F}\,\big(\mathsf{i} \wedge (((\mathsf{b} \vee \mathsf{c} \vee \mathsf{k}) \Rightarrow \mathsf{F}\,(\mathsf{j} \wedge \mathsf{F}\,\mathsf{i})) \;\mathsf{U}\; ((\mathsf{l} \vee \mathsf{f}) \wedge (\neg(\mathsf{b} \vee \mathsf{c} \vee \mathsf{d}) \;\mathsf{U}\; \mathsf{g})))\big),$$

which encompasses 11 unique atomic propositions and a complex composition of different LTL operators. The corresponding LDBA consists of a large number of 656 states. However, by explicitly reasoning about the different ways of satisfying the specification, DeepLTL achieves an average success rate of 98% (across 5 different random seeds).

At the same time, we note that even seemingly simple formulae can be difficult to accomplish. In *LetterWorld*, this is especially true if they involve a large number of propositions to avoid, such as the specification $\neg(\mathsf{a} \vee \mathsf{b} \vee \mathsf{c} \vee \mathsf{d} \vee \mathsf{e}) \;\mathsf{U}\; (\mathsf{f} \wedge \mathsf{F}\,\mathsf{g})$, on which DeepLTL achieves a comparatively low

Table 8: Experimental comparison to RAD-embeddings on different finite-horizon tasks in the *ZoneEnv* environment. The numbers in parentheses represent the number of states in the LDBA/DFA. We report the *success rate* (SR) and average number of steps to satisfy the task (μ). Results are averaged over 5 seeds and 500 episodes per seed. "$\pm$" indicates the standard deviation over seeds.

	RAD-embeddings		DeepLTL	
	SR (↑)	μ (↓)	SR (↑)	μ (↓)
φ_6 (6)	$\mathbf{0.94}_{\pm 0.06}$	$469.72_{\pm 59.99}$	$0.92_{\pm 0.06}$	$\mathbf{303.38}_{\pm 19.43}$
φ_7 (6)	$0.88_{\pm 0.06}$	$474.34_{\pm 62.14}$	$\mathbf{0.91}_{\pm 0.03}$	$\mathbf{299.95}_{\pm 09.47}$
φ_8 (8)	$\mathbf{0.96}_{\pm 0.06}$	$399.38_{\pm 68.63}$	$\mathbf{0.96}_{\pm 0.04}$	$\mathbf{259.75}_{\pm 08.07}$
φ_9 (5)	$\mathbf{0.92}_{\pm 0.02}$	$304.88_{\pm 38.42}$	$0.90_{\pm 0.03}$	$\mathbf{203.36}_{\pm 14.97}$
φ_{10} (4)	$0.87_{\pm 0.11}$	$336.90_{\pm 88.32}$	$\mathbf{0.91}_{\pm 0.02}$	$\mathbf{187.13}_{\pm 10.61}$
φ_{11} (4)	$\mathbf{0.99}_{\pm 0.01}$	$149.02_{\pm 19.72}$	$0.98_{\pm 0.01}$	$\mathbf{106.21}_{\pm 07.88}$
reach/avoid (4/5)	$0.91_{\pm 0.04}$	$401.12_{\pm 33.92}$	$\mathbf{0.93}_{\pm 0.02}$	$\mathbf{261.97}_{\pm 11.79}$
parity (5)	$\mathbf{0.98}_{\pm 0.01}$	$292.21_{\pm 27.27}$	$0.96_{\pm 0.03}$	$\mathbf{211.07}_{\pm 12.16}$
large1 (15)	$0.65_{\pm 0.32}$	$550.51_{\pm 95.79}$	$\mathbf{0.92}_{\pm 0.06}$	$\mathbf{314.72}_{\pm 23.23}$
large2 (19)	$0.66_{\pm 0.31}$	$490.22_{\pm 40.23}$	$\mathbf{0.94}_{\pm 0.02}$	$\mathbf{322.38}_{\pm 24.02}$

success rate of 82% across 5 random seeds. In general, whether a given formula is challenging to satisfy depends primarily on the dynamics of the underlying MDP.

Finally, we note that our approach relies on being able to construct an LDBA for a given specification, which is doubly-exponential in the size of the formula in the worst case (Sickert et al., 2016). Inevitably, there thus exist LTL specifications to which our approach is not applicable, since it is infeasible to construct the corresponding LDBA.

G.8 Comparison to RAD-embeddings

Table 8 lists experimental results of DeepLTL and RAD-embeddings (Yalcinkaya et al., 2024) on finite-horizon tasks in the *ZoneEnv* environment. We consider the tasks φ_6 — φ_{11} from Table 2, *reach/avoid* tasks (see Section 5.1), as well as a *parity* task, and two tasks with large associated automata.[4][5] The *parity* task specifies that the agent has to reach blue while visiting green an even number of times, and yellow while visiting magenta an even number of times. Note that this task is not expressible in LTL, but our method can still be applied to the corresponding LDBA.

For RAD-embeddings, we use frozen GATv2 embeddings pre-trained on compositional RAD DFAs, and the policy is also trained on compositional RAD DFAs for 15M environment interaction steps. Throughout the experiments, we use the hyperparameters provided in (Yalcinkaya et al., 2024).

The results shows that RAD-embeddings and DeepLTL achieve similar success rates, but DeepLTL generally requires significantly fewer steps until task completion. They also demonstrate that our approach indeed generalises to tasks not expressible in LTL, such as *parity*. Finally, we note that our method achieves substantially higher success rates on tasks with an increased number of states in the corresponding automaton (*large1/2*). These results further support the advantages of our approach of combining a high-level reasoning step with learned embeddings of reach-avoid sequences.

G.9 Trajectory visualisations

We qualitatively confirm that DeepLTL produces the desired behaviour by visualising trajectories in the *ZoneEnv* and *FlatWorld* environments for a variety of tasks (Figures 12 and 13).

[4] *large1* corresponds to the formula $((\text{green} \Rightarrow (\neg\text{blue U } ((\text{magenta} \wedge \text{F blue}) \wedge (\neg\text{green U blue})))) \text{ U } (\text{yellow} \wedge \text{F } (\text{magenta} \wedge \text{F blue})))$.

[5] The formula for *large2* is $(\text{green} \Rightarrow \text{F } (\text{yellow} \wedge (\text{F } (\text{magenta} \wedge \text{F blue})))) \text{ U } ((\text{blue} \wedge \text{F magenta}) \wedge (\neg\text{yellow U green}))$.

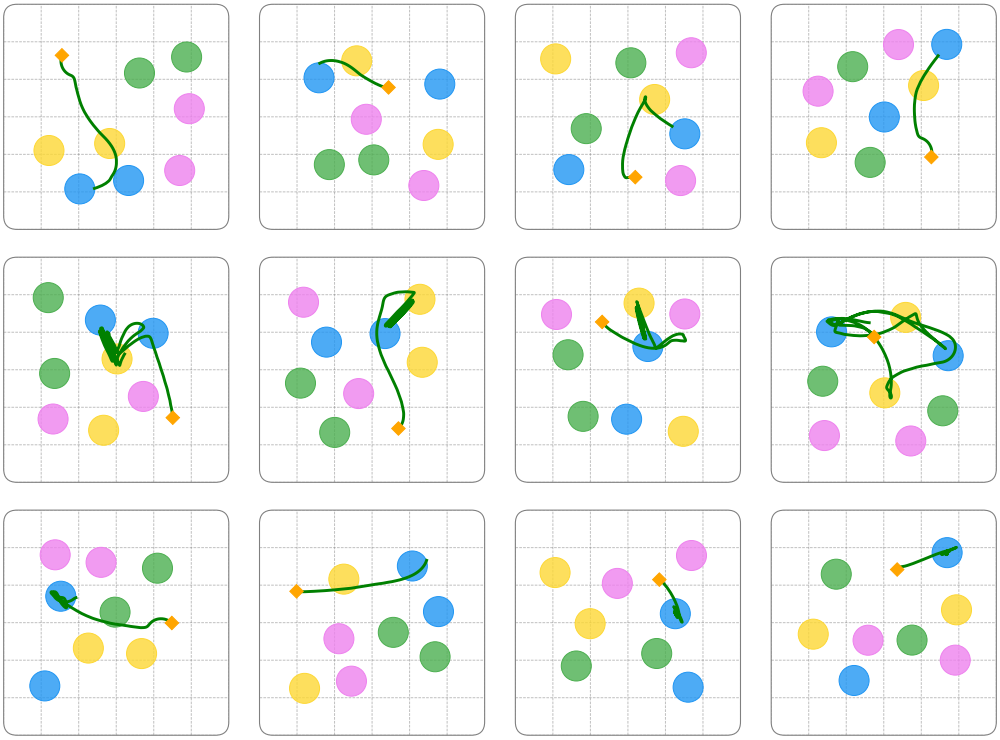


Figure 12: Example trajectories of DeepLTL in the *ZoneEnv* environment. (Top row) Trajectories for the task $\mathsf{F}\,(\mathsf{yellow} \wedge (\neg\mathsf{green}\ \mathsf{U}\ \mathsf{blue}))$. (Middle row) Trajectories for the task $\mathsf{G}\,\mathsf{F}\,\mathsf{yellow} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{blue} \wedge \mathsf{G}\,\neg\mathsf{green}$. (Bottom row) Trajectories for the task $\mathsf{F}\,\mathsf{G}\,\mathsf{blue} \wedge \mathsf{G}\,\neg\mathsf{magenta}$.

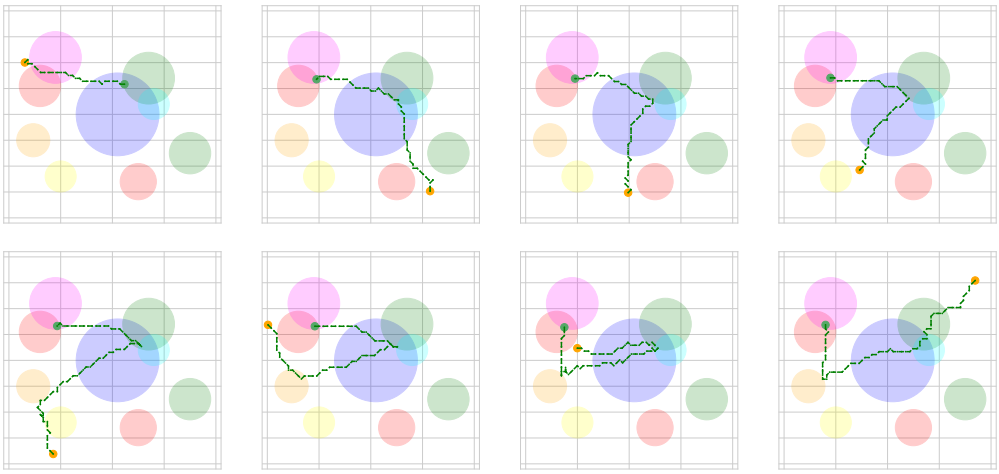


Figure 13: Example trajectories of DeepLTL in the *FlatWorld* environment. (Top row) Trajectories for the task $\mathsf{F}\,(\mathsf{red} \wedge \mathsf{magenta}) \wedge \mathsf{F}\,(\mathsf{blue} \wedge \mathsf{green})$. (Bottom row) Trajectories for the task $(\neg\mathsf{red}\ \mathsf{U}\ (\mathsf{green} \wedge \mathsf{blue} \wedge \mathsf{aqua})) \wedge \mathsf{F}\,(\mathsf{orange} \wedge (\mathsf{F}\,(\mathsf{red} \wedge \mathsf{magenta})))$.