Semantically Labelled Automata for Multi-Task Reinforcement Learning with LTL Instructions*

Alessandro Abate[1], **Giuseppe De Giacomo**[1], **Mathias Jackermeier**[1], **Jan Kretínský**[2,3], **Maximilian Prokop**[2,3] and **Christoph Weinhuber**[1]
[1] University of Oxford
[2] Technical University of Munich
[3] Masaryk University Brno

Abstract

We study multi-task reinforcement learning (RL), a setting in which an agent learns a single, universal policy capable of generalising to arbitrary, possibly unseen tasks. We consider tasks specified as linear temporal logic (LTL) formulae, which are commonly used in formal methods to specify properties of systems, and have recently been successfully adopted in RL. In this setting, we present a novel task embedding technique leveraging a new generation of *semantic* LTL-to-automata translations, originally developed for temporal synthesis. The resulting *semantically labelled* automata contain rich, structured information in each state that allow us to (i) compute the automaton efficiently on-the-fly, (ii) extract expressive task embeddings used to condition the policy, and (iii) naturally support full LTL. Experimental results in a variety of domains demonstrate that our approach achieves state-of-the-art performance and is able to scale to complex specifications where existing methods fail.

1 Introduction

Developing reinforcement learning (RL) agents capable of executing complex, temporally extended tasks, such as those expressed in Linear Temporal Logic (LTL) [Pnueli, 1977], is an active area of research in artificial intelligence. A long line of existing research investigates training RL agents to execute a single LTL instruction [Sadigh *et al.*, 2014; Brázdil *et al.*, 2014; Hahn *et al.*, 2019; Hasanbeig *et al.*, 2022; Voloshin *et al.*, 2023]. Traditionally, the inherent memory requirement of LTL specifications is addressed by constructing an automaton (typically a variant of a Büchi automaton) corresponding to the LTL specification. However, this couples the resulting policy to that specific LTL formula. Consequently, even minor changes in the objective, e.g. altering the items that a warehouse robot needs to collect, requires retraining the agent. Thus, single-task approaches are ineffective at agent deployment time, where retraining is infeasible, or when task specifications change frequently.

As a result, recent years have seen increased interest in the *multi-task* RL setting [Oh *et al.*, 2017]. Generally speaking, multi-task RL focuses on training a single, generalist policy capable of executing arbitrary tasks that may not be known apriori. Intuitively, such a policy learns a set of reusable behaviours and how to compose them in order to achieve new tasks. This is particularly challenging for LTL tasks, as the agent must additionally manage the required memory.

Recent approaches to tackle multi-task RL with LTL specifications can be broadly categorised into *decomposition*-based methods [León *et al.*, 2022; Qiu *et al.*, 2023; Guo *et al.*, 2025], which break down the objective into several subtasks, and *holistic* approaches that consider the full specification [Vaezipoor *et al.*, 2021; Xu and Fekri, 2024; Jackermeier and Abate, 2025]. While decomposition into subtasks facilitates generalisation by allowing the agent to recombine learned subtask policies, it may lead to suboptimal solutions due to missing context. For instance, given the task "approach shelf, then pick up hammer", a warehouse robot focusing only on the first objective is unaware of which shelf to approach. To avoid these kinds of problems, holistic approaches seek policies that consider the full context of the task. This, however, is significantly more challenging, as the generalisation must be achieved solely through learning, which necessitates an amenable task embedding. In particular, arbitrary tasks must be embedded in a finite-dimensional feature space that preserves the semantic similarity of tasks, while also effectively encoding the necessary memory.

Our contribution. In this paper, we propose novel task embeddings based on a new generation of LTL-to-automata translations [Esparza and Křetínský, 2014; Esparza *et al.*, 2020]. Unlike traditional translations, these approaches are *semantic*: each automaton state is annotated with additional information (so-called *semantic labelling*) that intuitively describes the semantic meaning of that state. Recent work already exploited semantic labelling successfully in challenging problems from formal methods such as parity game solving [Křetínský *et al.*, 2019; Křetínský *et al.*, 2023] and reactive synthesis [Křetínský *et al.*, 2025]. In this work, we demonstrate that semantic labelling enables an effective embedding of arbitrary LTL tasks, while naturally addressing their memory requirements. In contrast to previous methods, our approach is both (i) applicable to full LTL and (ii) lightweight, as it can be computed efficiently on the

*For full version and appendicies see [Abate *et al.*, 2026]

fly. Finally, we design a policy around these embeddings and demonstrate that this enables successful generalisation to specifications of previously infeasible complexity.

2 Preliminaries

Linear temporal logic. Let AP be a finite set of atomic propositions. LTL formulas are generated by $\varphi ::= tt \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$. where $a \in \mathsf{AP}$, and $\mathbf{X}$ ("Next"), $\mathbf{U}$ ("Until") are temporal operators. Further, we define the common abbreviations , $\mathbf{F}\,\varphi \equiv tt\,\mathbf{U}\,\varphi$ ("Finally/Eventually") and $\mathbf{G}\,\varphi \equiv \neg\,\mathbf{F}\,\neg\varphi$ ("Globally/Always") [Pnueli, 1977].

An *ω-word* w is an infinite sequence of letters $w[0]\,w[1]\,w[2]\ldots$ with $w[i] \in \Sigma = 2^{\mathsf{AP}}$. We denote the infinite suffix $w[i]\,w[i+1]\ldots$ by w_i and present the full satisfaction relation $w \models \varphi$ in Appendix A.1. The *language* of φ, is the set of all satisfying words, denoted by $[\varphi]$. LTL has several fragments, most prominently *Safety* and *Guarantee*. Intuitively, they denote that something bad never happens and that something good happens eventually, respectively. Further, we consider *recurrence* (a guarantee holds infinitely often) and *persistence* (a safety formula holds eventually).

Limit-deterministic Büchi automata. We define a limit-deterministic Büchi automaton (LDBA) as a tuple $\mathcal{B} = (Q, \Sigma, \delta, \mathcal{E}, q_0, F)$, where Q is a finite set of states, $\Sigma := 2^{\mathsf{AP}}$ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $\mathcal{E} : Q \to 2^Q$ are (non-deterministic) epsilon transitions, $q_0 \in Q$ the initial state, and $F \subseteq Q$ is a set of accepting states, satisfying the following. There exists a partitioning $Q = Q_I \uplus Q_A$ such that (i) the transition function never changes the component, (ii) epsilon transitions only exist from Q_I to Q_A, (iii) every accepting state is in Q_A, and (iv) the initial state is in Q_I.

A run on an ω-word $\omega = \sigma_0\sigma_1\cdots$ is an infinite sequence of states $q_0q_1\cdots$ that begins in the initial state and evolves according to the transition function except for at most one index where it may take an ϵ-transition to Q_A and remain in Q_A thereafter. Let j be the index of the epsilon transition. Then, formally, for all $i \geq 0$ with $i \neq j$, $q_{i+1} = \delta(q_i, \sigma_i)$ and $q_j \in Q_I, q_{j+1} \in \mathcal{E}(q_j)$. A run is accepting if it visits F infinitely often and a word is accepted by the LDBA if there exists at least one accepting run. For any LTL formula φ there exists an LDBA that accepts $[\varphi]$, e.g. [Sickert *et al.*, 2016].

Markov decision processes. We define a Markov decision process (MDP) as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ the (unknown) transition kernel, $\mu \in \Delta(\mathcal{S})$ the initial state distribution, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function, and $\gamma \in [0, 1)$ the discount factor. Here $\Delta(X)$ denotes the set of probability distributions over set X and we write $x \sim \mu$ to denote that x is sampled from a distribution μ. A (memoryless) policy is a map $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, from state s to a distribution over actions. Rolling out π in $\mathcal{M}$ yields a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ with $s_0 \sim \mu$, $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim \mathcal{P}(s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$. A policy conditioned on task φ is denoted as $\pi \mid \varphi$ or $\pi(\cdot \mid s_t, \varphi)$ when applied to a concrete state s_t. We interpret MDPs via a labelling map $\mathcal{L} : \mathcal{S} \to 2^{\mathsf{AP}}$. Given a trajectory τ, $\mathcal{L}$ induces an ω-word $w = \mathcal{L}(s_0)\mathcal{L}(s_1)\cdots \in (2^{\mathsf{AP}})^\omega$, on which we evaluate LTL .

Reinforcement learning. Following [Sutton and Barto, 2018], the objective of reinforcement learning (RL) is to find a policy $\pi^\star$ that maximises the *expected discounted return* $J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t\, r_t]$. We write $\tau \sim \pi$ to emphasise that the trajectory distribution depends on π. The *state-value function* of a policy π is $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$ and corresponds to the expected discounted return starting from state s and following policy π. In addition, we also consider a specification-driven objective, namely maximizing the probability of satisfying an LTL task φ. For a policy π, we define this satisfaction probability as $\Pr(\pi \models \varphi) = \mathbb{E}_{\tau\sim\pi}\big[\mathbb{1}[\tau \models \varphi]\big]$, where $\mathbb{1}[\tau \models \varphi]$ is 1 if the trace induced by τ satisfies φ, and 0 otherwise.

3 Problem Definition

We study multi-task RL with LTL specifications, following [Vaezipoor *et al.*, 2021; Jackermeier and Abate, 2025]. In contrast to single-task RL, we consider *task-conditioned* policies $\pi \mid \varphi$ that receive a *current* task φ as input. Such policies can adapt to a new task φ without requiring any retraining. In particular, the MDP is inacessible and solely φ itself may be subject to any computation before and during the execution. Formally, the goal is to learn a single universal policy that maximises the overall satisfaction probability over some distribution $\mathcal{D}$ over LTL tasks:

$$\pi^\star \in \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\varphi\sim\mathcal{D},\\ \tau\sim\pi\mid\varphi}} [\mathbb{1}[\tau \models \varphi]] \tag{1}$$

The following two key challenges arise in this setting.

Memory requirements of LTL tasks. Notice that LTL tasks are inherently *non-Markovian*: the agent needs to be aware of already observed labels to decide its next action, as exemplified by our running example $\mathbf{F}\,r \wedge \mathbf{FG}\,y$. In the same MDP state, the agent has to perform different actions depending on whether it has already seen r; if it has, it can focus on continuously achieving y; otherwise, it needs to satisfy the finite goal $\mathbf{F}\,r$ first. While it is in principle possible to learn a non-Markovian policy $\pi(\cdot \mid s_0, \ldots, s_t, \varphi)$ conditioned on the entire trajectory history, this introduces significant complexity in practice. Hence most RL approaches with LTL tasks aim to recover a Markovian view by augmenting the MDP state with memory (typically the states of an automaton for φ) that keeps track of the current task's "progress" (intuitively, tracking the already achieved subgoals of φ).

Universal progress interface. In single-task RL, tracking progress of subgoals and conditioning a policy on it is straightforward. The common approach is to construct an automaton for the given LTL task, as by design, the automaton states form sufficient memory for deciding what to do next in the given task. We then retrieve a Markovian setting by training a separate sub policy for each state. At test time, the conditioning on task progress is achieved by tracking the current automaton state and employing the respective sub-policy.

However, in multi-task RL, conditioning via state-dependent sub-policies is no longer feasible, since the automaton is not fixed a priori, but changes with every new task. Instead, a universal policy capable of being conditioned

on *any* possible task needs to be capable of processing *any* possible automaton. This necessitates a *universal progress interface* (UPI) which is a finite-dimensional embedding of arbitrary tasks, that the policy can be conditioned on. A UPI should (i) reduce any tasks to a common, learnable representation (ii) update that representation depending on achieved progress, and (iii) be computationally cheap, as at test time, it needs to be computed online during interaction with the environment. To achieve the task-conditioned behaviour of the policy, the UPI needs to be semantically meaningful: if two different tasks require similar behaviour, they should have similar embeddings. For instance, $\mathbf{F}\, r \wedge \mathbf{FG}\, y$ and $\mathbf{F}\, r$ should have similar embeddings as both require an r to proceed.

4 Existing Approaches

We proceed to discuss two current approaches to constructing a UPI which we build upon.

4.1 Formula Progression Approach

In LTL2ACTION [Vaezipoor *et al.*, 2021], the LTL formula itself is embedded as the universal progress interface. Crucially, task progress can be updated by so-called *formula progression* [Bacchus and Kabanza, 1996]. Intuitively, for a formula φ_t and a letter σ the progressed formula $\mathit{prog}(\sigma, \varphi_t)$ captures what remains to be satisfied after seeing σ. For example, if $\varphi_t = \mathbf{F}(r \wedge \mathbf{FG}\, y)$ and $\sigma = \{r\}$, then $\mathit{prog}(\sigma, \varphi_t) = \mathbf{FG}\, y$ (see Appendix A.2 for a full definition).

LTL2ACTION then embeds the current (progressed) formula via a graph neural network (GNN) [Scarselli *et al.*, 2009] g that operates on LTL syntax trees. The GNN is trained end-to-end with the task-conditioned policy $\pi(\cdot \mid s_t, g(\varphi_t))$ via RL by issuing a reward of 1 exactly when φ_t progressed to tt, which denotes satisfaction of the task.

A benefit of this approach is that formula progression can be computed efficiently and on the fly, i.e. each step is only computed when it is actually required. However, its reliance on eventually progressing to tt to give rewards limits this approach to the co-safety (guarantee) fragment of LTL. Giving rewards for general LTL tasks requires tracking additional information which plain formula progression does not provide.

4.2 Automata-Theoretic Approach

The common approach to support full LTL is to convert the current task to an automaton and keep track of the current automaton state as a UPI. This is the approach taken by DEEPLTL [Jackermeier and Abate, 2025], which employs LDBAs as suitable automata. Formally, the task-conditioned policy $\pi \mid \varphi$ then operates over the *product MDP* $\mathcal{M}_\varphi$, whose state space consists of the MDP state space $\mathcal{S}$ augmented with the LDBA state space $\mathcal{Q}$. Non-deterministic ε-transitions are handled by introducing special ε-actions $\varepsilon_{q'}$ that update the current LDBA state q to $q' \in \mathcal{E}(q)$ without modifying the MDP state. Assigning positive rewards of 1 to visiting accepting states F in the LDBA then enables learning a task-conditioned policy. Formally, the policy is optimised via the following objective, where $\mathbb{1}_F$ is the indicator function of F:

$$\pi^\star(\cdot \mid s_t, q_t) \in \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\varphi \sim \mathcal{D}, \\ \tau \sim \pi \mid \varphi}} \Big[\sum_{t'=0}^{\infty} \gamma^t \mathbb{1}_F(q_{t'}) \Big]. \tag{2}$$

The key question in the automata-theoretic approach is how to obtain meaningful features of LDBA states from arbitrary formulae that are not known apriori. DEEPLTL proposes so-called *reach-avoid sequences* as a common representation of automaton states. Intuitively, these are sequences of letters to reach and avoid that would result in visiting accepting states infinitely often. They hence form a complete plan on how to produce a satisfying trace from the current state. To chose between the many sequences that could represent a state, DEEPLTL learns a critic network that judges the feasibility of any sequence in the given MDP. Formally, DEEPLTL learns a policy $\pi(\cdot \mid s_t, e(\varsigma))$ conditioned on a reach-avoid sequence ς embedded via an embedding network e. At test time, the agent tracks the current state of the LDBA and continually searches for the optimal (according to the critic) reach-avoid sequence $\varsigma^\star$ for representing the current state q_t, and executes actions given by $\pi(\cdot \mid s_t, e(\varsigma^\star))$.

While this approach works for full LTL, the search for reach-avoid sequences is prohibitively expensive as (i) the entire LDBA, needs to be constructed upfront and (ii) all of its accepting paths need to be exhaustively enumerated to find the optimal reach-avoid sequence (see Appendix B for an exact derivation of the complexity). This limits DEEPLTL to relatively simple tasks, where computing and enumerating sequences of the entire automaton is feasible.

Rationale for LDBAs. The choice of LDBA as automata is important as they combine several useful properties. First, their Büchi acceptance can be converted naturally to a reward signal s.t. maximizing rewards coincides with optimising satisfaction probability [Hahn *et al.*, 2019]. Crucially, this is not possible for other acceptance conditions like Rabin, and impractical for parity [Hahn *et al.*, 2022]. Note however, that this requires slightly more sophisticated objectives than the one we presented in Equation 2, in order to address the bias introduced by the discount factor γ [Hahn *et al.*, 2019; Bozkurt *et al.*, 2020; Voloshin *et al.*, 2023]. Further, while some non-determinism is unavoidable when aiming for a Büchi automaton that captures full LTL, the non-determinism of LDBAs is very manageable. Specifically, their non-determinism is *good-for-MDP*, meaning that it can be resolved by the agent (i.e. the epsilon transitions turn into new actions) without sacrificing any optimality of the policy.[1]

4.3 Other Related Work

Several other works recently addressed the problem of multi-task RL with LTL specifications. Many of these approaches decompose the LTL specification into sub-tasks, which are then completed one-by-one [León *et al.*, 2022; Qiu *et al.*, 2023; Liu *et al.*, 2024; Guo *et al.*, 2025]. However, this generally still requires constructing the entire automaton upfront and furthermore, can lead to *myopic* solutions due to missing context. In contrast, we specifically design our approach to not suffer from such suboptimality. Xu and Fekri [2024] propose a non-myopic approach based on the option framework [Sutton *et al.*, 1999], but this is limited to a small fragment of LTL. Recently, Giuri *et al.* [2025] investigated

[1] Technically, not every LDBA is good-for-MDPs, but the ones produced by [Sickert *et al.*, 2016; Esparza *et al.*, 2020] are.

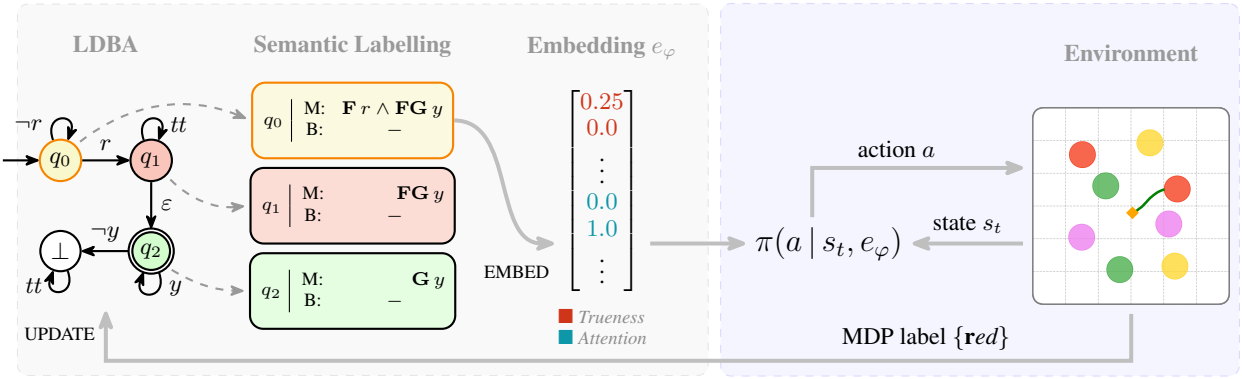

Figure 1: Schematic overview of our approach for the ZoneEnv environment (a robot navigating a 2D plane with zones of different colours) and the LTL task $\varphi = \mathbf{F}\, \mathbf{r}ed \wedge \mathbf{FG}\, \mathbf{y}ellow$. We first convert the task to an LDBA with semantic labelling (see Section 5.1). Note that initially only q_0 is constructed. Then the semantic labelling of the current state q_0 is embedded and passed to the policy (see Section 5.2). Together with the current state of the MDP s_t, the policy yields action a in the MDP. The robot's trajectory in the MDP (green line) enters a zone with MDP label $\mathbf{r}ed$ which is used to update the automaton. Only then, the corresponding successor q_1 is constructed in the automaton.

GNNs to improve the representation learning capabilities of DeepLTL, but their approach suffers from the same fundamental drawbacks as DeepLTL.

5 Semantically Labelled Automata Approach

We start by giving an overview of our approach. We follow the automata-theoretic approach in the sense that we construct an LDBA for the current task, update its state to track progress and condition our policy on the current state. The crux is to compute a meaningful embedding of any possible automaton state for any possible LTL formula (and ideally just by looking at that state alone, without considering the entire automaton). Since there are infinitely many automaton states and we can only observe finitely many in training, the key question is how to embed the states into a structured feature space amenable to learning, i.e. where semantically similar states are close to each other. Traditionally, states of (limit-)deterministic automata for LTL formulae bear no logical structure and are simply identified with a numeric index produced by the translation algorithm, so this may seem ambitious. However, we employ more modern, *semantic translations* to construct the LDBA, which are the product of a long line of research on constructing small automata [Křetínský and Esparza, 2012; Esparza and Křetínský, 2014; Esparza *et al.*, 2020]. The key property of these translations is that each state is identified with a simple structure (e.g. a pair or a vector) of LTL formulae. This so-called *semantic labelling* fully describes the semantics of a state (that is, which subgoals remain to be achieved from here on) and thus, embedding this information can inform the policy about progress of the task and remaining obligations. Crucially, the semantic labelling of a state is available without having to compute the entire automaton. Instead, required parts of the automaton can be computed on-demand, depending on the observed MDP labels. See Figure 1 for an illustration of this approach.

5.1 Semantic Labelling

The semantic labelling of our automaton states originates from the translation presented in [Esparza *et al.*, 2020], which is implemented in Owl [Křetínský *et al.*, 2018]. Their LDBAs require keeping track of only two LTL formulae, which is achieved through upfront normalisation [Sickert and Esparza, 2020]. First, we have the **M***ain formula*, which is the entire formula in the initial state and for every subsequent state is derived using formula progression. Intuitively, the main formula can be thought of as the "remaining language" from the current state onwards. However, for a recurrence formula like $\mathbf{GF}\, r$ (i.e. reach r infinitely often), the remaining obligation will never change and thus, just tracking the main formula cannot distinguish between words that satisfy this property and ones that do not. Thus, a second formula, the **B***reakpoint formula* is required to track the inner formula of recurring obligations ($\mathbf{F}\, r$ in this case). Whenever this formula progresses to tt, it resets and emits a Büchi acceptance signal. That way, seeing a Büchi signal infinitely often corresponds directly to satisfying the inner formula infinitely often.

As an example, consider the automaton of Figure 1. The main formula of q_0 is the entire task and once we observe r, the automaton updates to state q_1 in which the main formula has progressed to $\mathbf{FG}\, y$. The ε-transition guesses when the safety formula $\mathbf{G}\, y$ starts to hold and q_2 verifies that guess through formula progression. This example has no breakpoint formulae due to the lack of recurrence formulae; see Appendix C for an example with a breakpoint formula.

5.2 Embedding the Semantic Labelling

Embedding the semantic labelling requires capturing the relevant information of two LTL formulae, which we tackle individually and concatenate the result. Recall that the embedding space needs to create numerical proximity of semantically similar states (or rather similar formulae) to enable generalisation. Intuitively, the similarity of two formulae is proportional to how much their languages overlap. Hence, our features intuitively approximate a formula's language by probing the effects of letters when seen at the current- or subsequent steps. Additionally, we employ simple complexity measures of formulae (e.g. the height of the syntax tree as suggested in [Křetínský *et al.*, 2023]) to allow the agent to

choose less complex states when resolving ε-transitions.

Exploiting trueness. This feature approximates the immediate effect of observing a letter by measuring how much each letter progresses the formula towards or away from satisfaction. We thus need a measure of proximity to satisfaction. Intuitively, it is clear that $\mathbf{FG}\, y$ is closer to satisfaction than $\mathbf{F}\, r \wedge \mathbf{FG}\, y$, since the former already achieved a subgoal. To capture this, we resort to *trueness* which was first introduced in [Křetínský *et al.*, 2019]. Intuitively, trueness measures how easy it is to satisfy an LTL formula, by propositional approximation of the temporal behaviour. Formally, to compute the trueness $tr(\varphi)$ of a formula φ we replace all temporal subformulae ψ by new propositional variables x_ψ and compute the ratio of satisfying assignments to total assignments in the resulting propositional formula. For example for $\mathbf{F}\, r \wedge \mathbf{FG}\, y$, we replace all temporal subformulae, and get $x_{\mathbf{F}\, r} \wedge x_{\mathbf{FG}\, y}$. In the resulting conjunction, 1 out of 4 assignments satisfy the formula and thus $tr(\mathbf{F}\, r \wedge \mathbf{FG}\, y) = 0.25$. Further, we have $tr(\mathbf{FG}\, y) = 0.5$, which indicates that the latter formula is easier to satisfy and progressing to the latter is desirable.

We use trueness to compute a feature value for every MDP label of $\mathcal{L}$. As we are interested in the impact of a letter, we compare the change in trueness before and after seeing that letter, i.e. after progressing the formula using that letter. Formally, for a formula φ and a letter σ the trueness feature f_{tr} is defined as:

$$f_{tr}(\varphi, \sigma) = tr(prog(\varphi, \sigma)) - tr(\varphi).$$

In the embedding vector of Figure 1, the first two red numbers correspond to the feature values $f_{tr}(\varphi, \{r\})$ and $f_{tr}(\varphi, \{y\})$, where $\varphi = \mathbf{F}\, r \wedge \mathbf{FG}\, y$. By looking at these two numbers, the agent can infer that an $\{r\}$ would make progress while seeing $\{y\}$ currently has no effect. Symmetrically to progress, this feature also captures violations. E.g. in q_2 where the main formula is $\mathbf{G}\, y$ we have the feature value $f_{tr}(\mathbf{G}\, y, \{\}) = -0.5$, indicating that not seeing a y (terminally) hurts the progress towards satisfying $\mathbf{G}\, y$. We further refine this feature be employing different normalisations (e.g. among all letters, or among all positive/negative values) that result in slightly different semantic meanings. For a full list and explanation of these, we refer to Appendix D. Ultimately, despite being unable to capture deep temporal relations (e.g. $\mathbf{F}\, r \wedge \mathbf{G}\, \neg r$ having trueness 0.25 despite being unsatisfiable), trueness yields a great propositional approximation of task-progress, especially for formulae appearing in practice.

Propositional attention. Trueness alone yields a myopic embedding, meaning that it only considers the immediate effect of seeing a letter while disregarding what happens afterwards. For example, the trueness feature values of $\mathbf{F}\, r \wedge \mathbf{FG}\, y$ alone, do not yield any indication that after seeing $\{r\}$, $\{y\}$ will be important, which can result in suboptimal behaviour in certain environments. To combat this, we propose an attention-like feature, where we relate pairs of propositions by how reliant one is on the other for satisfying the formula.

Formally, for a formula φ and a pair of propositions (p, q), we first compute the formula after seeing $\{p\}$ denoted as φ'. Then, we compute the *obligation sets* [Li *et al.*, 2013] of φ', denoted by $ob(\varphi')$. A single obligation set is a letter $\sigma \in 2^{AP}$ that if seen infinitely often, satisfies the formula, i.e. $\sigma^\omega \models \varphi'$. We provide the full definition of obligation sets in Appendix A.3, but intuitively, the obligation sets approximate the infinite behaviours of φ' on a propositional level. To compute the feature value, we then count how often q appears in the obligation set (treating q and $\neg q$ separately) and divide by the total number of obligation sets. Putting it all together, the positive propositional attention feature f^+_{att} for a formula φ and a pair of propositions (p, q) is given by

$$f^+_{att}(\varphi, p, q) = \frac{|\{o \in ob(\varphi') \mid q \in o\}|}{|ob(\varphi')|},$$

where $\varphi' = prog(\varphi, \{p\})$ (and when $|ob(\varphi')| = 0$, the feature defaults to 0). Symmetrically, the negative propositional attention feature f^-_{att} counts the appearances of $\neg q$.

In the embedding vector from Figure 1, the two teal numbers correspond to the feature values $f^+_{att}(\varphi, r, r)$ and $f^+_{att}(\varphi, r, y)$, where $\varphi = \mathbf{F}\, r \wedge \mathbf{FG}\, y$. The first value being 0.0 indicates that after seeing $\{r\}$ once, it is completely irrelevant. Further, the second value being 1.0 indicates that after seeing $\{r\}$, seeing letters containing y is highly promising. Crucially, this information is already present in the embedding of q_0, allowing the agent to consider its future obligation of $\mathbf{FG}\, y$ while still primarily focussing on achieving $\mathbf{F}\, r$.

5.3 Policy Architecture

Our approach is realised in a deep RL setting. We next describe our policy architecture. Our policies take the current MDP state s and the semantic embedding of the current LDBA state q as input, and produce an action distribution over the MDP action space $\mathcal{A}$ and possible ε-actions. Recall that ε-actions $\varepsilon_{q'}$ (where $q' \in \mathcal{E}(q)$) correspond to following an ε-transition to q' while remaining in the same MDP state.

To handle this complex hybrid action space (note in particular that $\mathcal{A}$ may be continuous, and $|\mathcal{E}|$ is not known apriori), we propose a multi-headed neural network architecture for the policy. Given the current MDP state s and LDBA state q, we first compute latent representations of all LDBA states u in the ε-closure $\mathcal{E}(q) \cup \{q\}$ as follows: first, the MDP state s is encoded via a *state encoder*, which is either a multilayer perceptron (MLP) or a convolutional neural network (CNN), depending on the domain. Then, the semantic embedding of each LDBA state $u \in \mathcal{E}(q) \cup \{q\}$ is processed via a learnable linear projection layer. These feature vectors are concatenated and passed through a shared representation MLP for each LDBA state, producing latent embeddings $z_u \in \mathbb{R}^d$.

The latent embeddings are then fed into a linear *scoring* head $\lambda \colon \mathbb{R}^d \to \mathbb{R}$ that assigns logits to ε-actions, and the embedding of the current LDBA state q is additionally processed by an *environment actor* head π_{env} that outputs the parameters of a distribution over the MDP action space $\mathcal{A}$ (e.g. the mean and standard deviation of a Gaussian distribution). Intuitively, the scores $\lambda(z_u)$ model the probability of taking the ε-action that transitions to u, and $\lambda(z_q)$ corresponds to the probability of executing an MDP action rather than an ε-action. Formally, the policy induces the hybrid action distribution

$$\pi(a \mid s, q) = \begin{cases} \hat{\lambda}(z_u) & \text{if } a = \varepsilon_u, u \in \mathcal{E}(q) \\ \hat{\lambda}(z_q) \cdot \pi_{\text{env}}(a \mid z_q) & \text{if } a \in \mathcal{A}, \end{cases}$$

where ε_u is the ε-action that transitions to u and

$$\hat{\lambda}(z_v) = \frac{\exp(\lambda(z_v))}{\sum_{v' \in \mathcal{E}(q) \cup \{q\}} \exp(\lambda(z_{v'}))}$$

denotes the softmax-normalised score function. Note that if no ε-actions are available (i.e. $\mathcal{E}(q) = \emptyset$), then the action distribution reduces to simply $\pi_{\text{env}}(a \mid z_q)$.

Training procedure. We train the policy end-to-end via goal-conditioned RL [Liu *et al.*, 2022]: at the beginning of each episode, we sample a random LTL specification $\varphi \sim \mathcal{D}$ and assign a reward of 1 to visits to accepting states. To make training converge faster in practice, we follow the approach of Jackermeier and Abate (2025) and design a training curriculum consisting of different stages of increasingly challenging formulae. For example, we generally first train the agent with simple formulae of the form $\mathbf{F}\, a$ before sampling more complex tasks. Once the agent performs sufficiently well on tasks sampled from the current stage, we move on to the next stage. Our approach is independent of the underlying RL algorithm. Throughout our experiments, we use *proximal policy optimisation* (PPO) [Schulman *et al.*, 2017]. We provide further details on training, including on curricula, in Appendix E.

6 Experiments

We evaluate our approach, named SEMLTL, across a variety of domains and specifications.[2] We aim to answer the following research questions: **(RQ1)** Are our embeddings expressive enough to overcome the myopia of decomposition-based methods? **(RQ2)** Can SEMLTL successfully learn generalist policies for zero-shot executing arbitrary LTL instructions? **(RQ3)** Does our approach successfully scale to complex specifications, where previous methods fail?

RQ1. Consider the grid-world environment depicted in Figure 2 (left), inspired by [Vaezipoor *et al.*, 2021]. The agent, located on the left, receives one of the following two goals: $\mathbf{F}(\text{parcel} \wedge \mathbf{F}\,\text{hammer})$ or $\mathbf{F}(\text{parcel} \wedge \mathbf{F}\,\text{wrench})$, requiring it to enter the correct room via the one-way conveyor belts. For decomposition based approaches, the first subtask is $\mathbf{F}$ parcel in both cases. The agent thus has to choose a room (in order to pick up a parcel) without knowing the entire specification. Since both specifications are equally likely, but the agent has no way of differentiating between them, it can achieve a maximum success rate of 50%. In contrast, we observe that SEMLTL does not suffer from such myopia and quickly converges to the optimal policy (Figure 2, right).

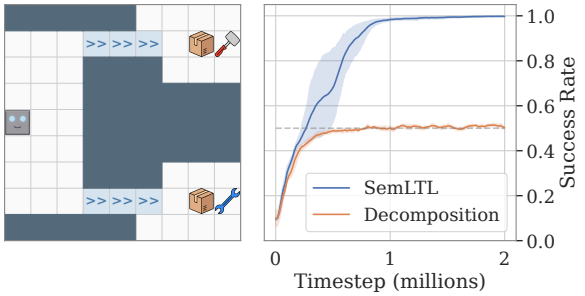

Figure 2: Conveyor environment example. The task is either $\mathbf{F}(\text{parcel} \wedge \mathbf{F}\,\text{hammer})$ or $\mathbf{F}(\text{parcel} \wedge \mathbf{F}\,\text{wrench})$. Once the agent is on a conveyor belt, it cannot go back. Without our propositional attention feature we can only achieve 50% (like other decomposition based approaches), whereas our full approach finds the optimal solution.

RQ2 and RQ3

We now proceed to conduct a large-scale experimental study to investigate RQ2 and RQ3.

Environments. We rely on established benchmarks to evaluate SEMLTL: the *LetterWorld* environment [Vaezipoor *et al.*, 2021], which consists of a 7×7 grid with randomly placed letters corresponding to atomic propositions, and *ZoneEnv* [Vaezipoor *et al.*, 2021], a high-dimensional robotic navigation environment with a continuous action space, where randomly placed zones of different colours form the atomic propositions. We consider a more difficult variant of ZoneEnv with 8 colours (instead of 4) and a more complex observation space based on an RGB lidar sensor. For further details, see Appendix F.1.

Baselines. We compare our approach to LTL2ACTION, the only other approach that does not have to compute the entire automaton upfront, and DEEPLTL, the state-of-the-art, automata-conditioned approach that does not suffer from myopia and is applicable to full LTL. See Appendix F for implementation details and hyperparameters of SEMLTL and the baselines, including training curricula.

Tasks. We consider a wide range of tasks for our evaluation. First, we compare to a set of established LTL tasks from the literature [Jackermeier and Abate, 2025], which we aggregate and refer to as SMALL in our results. Since these tasks are of limited complexity, we furthermore introduce several parameterised *families* of both finite- and infinite-horizon tasks that enable us to systematically assess the scalability of our approach. The LOCAL-SAFETY$[k, m]$ family ("avoidance" in [Vaezipoor *et al.*, 2021]), consists of m disjunctions of reach-avoid specifications $\varphi[k]$ where $\varphi[k] \equiv \neg a_1\ \mathbf{U}\ (r_1 \wedge (\neg a_2\ \mathbf{U}\ (r_2 \wedge \cdots \wedge (\neg a_k\ \mathbf{U}\ r_k))))$. Further, we define the variant GLOBAL-SAFETY$[k, m]$, which fixes a single global proposition a to avoid throughout the entire task. We also introduce the FINITE-REACTIVE$[k, m]$ family, where a simple finite-goal needs to be reached. However, whenever a trigger variable t is observed, the agent needs to react by achieving specified response variables. The overall task consists of k such reaction rules, where each permits m response variables. An example is $((t \rightarrow \mathbf{F}(r_1 \vee r_2))\ \mathbf{U}\ g$.

For infinite-horizon tasks, we first consider the COMPLEX-PATROL$[k, m]$ family. This consists of a disjunction of sequential reach tasks $\psi[k] \equiv \mathbf{F}(r_1 \wedge \mathbf{F}(\ldots \wedge \mathbf{F}\, r_k))$ that need to be satisfied recurrently while avoiding a global safety constraint. We also investigate the REACH-STAY$[k]$ family of the form $\psi[k] \wedge \mathbf{FG}\, a$, where a is an eventual safety target. Note that we do not consider this task family in LetterWorld, since the agent has to move at every timestep, and hence cannot stay continuously at the same proposition. These two families cover complex tasks of the persistence and recurrence frag-

[2] Code available at: github.com/mathiasj33/semltl

		Task Specification Type	$\lvert\mathcal{Q}\rvert$	$\lvert\delta\rvert$	LTL2ACTION	DEEPLTL	**SEMLTL (Ours)**	
					SR / μ_{acc}	SR / μ_{acc}	SR / μ_{acc}	μ_{states}
Finite-horizon	Letter	SMALL	9	23	$0.67_{\pm 0.12}$	$\mathbf{0.98}_{\pm 0.01}$	$\mathbf{0.98}_{\pm 0.01}$	$4.12_{\pm 0.02}$
		LOCAL-SAFETY[3, 3]	233	1,372	$0.57_{\pm 0.07}$	**TO**	$\mathbf{0.99}_{\pm 0.00}$	$5.38_{\pm 0.00}$
		GLOBAL-SAFETY[4, 6]	1,072	6,414	$0.12_{\pm 0.03}$	**TO**	$\mathbf{0.91}_{\pm 0.01}$	$6.77_{\pm 0.04}$
		FINITE-REACTIVE[8, 2]	300	2,115	$0.74_{\pm 0.04}$	**TO**	$\mathbf{1.00}_{\pm 0.00}$	$2.80_{\pm 0.10}$
	Zones	SMALL	9	23	$0.52_{\pm 0.07}$	$\mathbf{0.93}_{\pm 0.02}$	$0.92_{\pm 0.02}$	$3.40_{\pm 0.01}$
		LOCAL-SAFETY[3, 3]	83	379	$0.66_{\pm 0.11}$	$\mathbf{0.98}_{\pm 0.01}$	$0.93_{\pm 0.06}$	$5.09_{\pm 0.14}$
		GLOBAL-SAFETY[4, 8]	1,162	6,510	$0.39_{\pm 0.05}$	**TO**	$\mathbf{0.79}_{\pm 0.06}$	$6.73_{\pm 0.18}$
		FINITE-REACTIVE[8, 2]	107	619	$0.74_{\pm 0.11}$	**TO**	$\mathbf{0.98}_{\pm 0.00}$	$2.96_{\pm 0.16}$
Infinite-horizon	Letter	SMALL	7	16	n/a	$6.15_{\pm 1.97}$	$\mathbf{7.13}_{\pm 0.51}$	$4.76_{\pm 0.09}$
		COMPLEX-PATROL[5, 5]	49	227	n/a	$1.78_{\pm 0.26}$	$\mathbf{1.83}_{\pm 0.10}$	$6.44_{\pm 0.03}$
		ALWAYS-REACTIVE[5, 1]	38	204	n/a	**TO**	$\mathbf{3.06}_{\pm 0.07}$	$7.06_{\pm 0.02}$
	Zones	SMALL	4	10	n/a	$14.50_{\pm 6.71}$	$\mathbf{18.65}_{\pm 6.02}$	$3.59_{\pm 0.04}$
		REACH-STAY[5]	34	116	n/a	$914_{\pm 862}$	$\mathbf{2,503}_{\pm 497}$	$7.29_{\pm 0.15}$
		COMPLEX-PATROL[5, 5]	45	201	n/a	$2.13_{\pm 1.45}$	$\mathbf{2.44}_{\pm 0.44}$	$6.33_{\pm 0.10}$
		ALWAYS-REACTIVE[5, 1]	38	204	n/a	**TO**	$\mathbf{3.73}_{\pm 0.80}$	$7.00_{\pm 0.00}$

Table 1: Evaluation results of SEMLTL and the baselines. For each task family, $\lvert\mathcal{Q}\rvert$ denotes the average number of LDBA states, and $\lvert\delta\rvert$ the average number of transitions. We report success rate (SR) for finite tasks and number of visits to accepting states (μ_{acc}) for infinite tasks, as well as number of constructed LDBA states (μ_{states}) for SEMLTL. "**TO**" denotes failure to produce a single action within a time limit of 600s. Results are averaged over 5 seeds and 500 episodes per seed. SEMLTL demonstrates strong performance and robust scalability across tasks.

ment, respectively. We furthermore introduce the ALWAYS-REACTIVE$[k, m]$ family, where tasks require maintaining reactive responses over an infinite horizon. Concretely, a trigger t_0 must be seen infinitely often ($\mathbf{GF}\, t_0$), while responding to a set of k global reaction rules with m response variables each (e.g. $\mathbf{G}(t_0 \rightarrow \mathbf{F}(r_1 \vee t_1))$, where in particular t_1 could be another trigger). We choose suitable parameters to cover a broad spectrum of automaton sizes, and randomly sample 5 tasks per family for our experiments.

Results. Table 1 lists evaluation results of SEMLTL and the baselines on both evaluation environments. As is standard, we report success rates (SR) for finite-horizon tasks and average visits to accepting states (μ_{acc}) for infinite-horizon tasks. Intuitively, μ_{acc} captures how often the policy successfully completes an accepting cycle in the LDBA. We also report the number of LDBA states constructed by SEMLTL (μ_{states}).

We observe that SEMLTL in both environments successfully learns a policy that is able to generalise to diverse LTL specifications, achieving strong results throughout (**RQ2**). SEMLTL consistently performs much better than LTL2ACTION, which demonstrates the structure and learnability of our semantic embeddings. For finite tasks, the semantic labelling of LDBAs only consists of the main formula. Both approaches thus only track one formula (using formula progression) and differ solely in the way they embed that formula. We clearly observe that SEMLTL is able to leverage the same information much more effectively, and over a diverse set of typical RL specifications. DEEPLTL also achieves strong results on tasks where constructing the entire automaton and enumerating reach-avoid sequences is feasible. This is expected, since it exhaustively analyses the entire automaton to form a complete and optimal plan to inform its policy. However, for more complex specifications DEEPLTL fails to compute even a single action within the 600s time limit (denoted by **TO**), due to the prohibitively expensive sequence enumeration. In contrast, we observe that SEMLTL successfully scales to highly complex formulae and achieves strong performance even on large task instances with hundreds of automaton states (**RQ3**). Notably, SEMLTL often only considers a small fraction of the overall LDBA (see μ_{states} compared to $|Q|$). This highlights the suitability of many common RL tasks for on-the-fly automata construction, and demonstrates that the theoretical benefits of SEMLTL lead to significant advantages in practice. We provide full experimental results, including on more parameters for the different task families, per-specification results on SMALL tasks, and visualisations of SEMLTL trajectories in Appendix F.

7 Conclusion

We presented SEMLTL, a novel approach to train generalist policies for satisfying arbitrary LTL specifications. Our method is based on recent semantic LTL-to-automata translations that annotate each automaton state with rich, semantic information. Not only does this naturally enable support for full LTL, but it also allows automata states to be computed efficiently on the fly, without constructing the entire automaton upfront. We introduced a structured way to embed the semantic state information, and designed a suitable policy architecture based on these embeddings. Experimental results on a wide variety of specifications demonstrate that our approach successfully learns policies that can zero-shot generalise to novel LTL tasks, while being significantly more scalable and constructing only a fraction of the automaton.

Future work will extend our approach to real-life robotics settings with unknown labelling functions and enhance scalability via the recently proposed LTLf+/PPLTL+ temporal logics [Aminof *et al.*, 2025; De Giacomo *et al.*, 2025] and exploit good-for-MDP reduction pipelines [Weinhuber *et al.*, 2026].

Acknowledgements

This work was supported in parts by the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI (No. EP/Y028872/1). This work is part of the Intelligence-Oriented Verification&Controller Synthesis (InOVationCS) project funded by the European Union under Grant Agreement No. 101171844. MJ is funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1).

References

[Abate *et al.*, 2026] Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Kretínský, Maximilian Prokop, and Christoph Weinhuber. Semantically labelled automata for multi-task reinforcement learning with ltl instructions, 2026.

[Aminof *et al.*, 2025] Benjamin Aminof, Giuseppe De Giacomo, Sasha Rubin, and Moshe Y. Vardi. Ltlf+ and PPLTL+: extending ltlf and PPLTL to infinite traces. In *IJCAI*, pages 8447–8455. ijcai.org, 2025.

[Bacchus and Kabanza, 1996] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *AAAI/IAAI, Vol. 2*, pages 1215–1222. AAAI Press / The MIT Press, 1996.

[Bozkurt *et al.*, 2020] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *ICRA*, pages 10349–10355. IEEE, 2020.

[Brázdil *et al.*, 2014] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014.

[De Giacomo *et al.*, 2025] Giuseppe De Giacomo, Yong Li, Sven Schewe, Christoph Weinhuber, and Pian Yu. Solving mdps with ltlf+ and PPLTL+ temporal objectives. In *IJCAI*, pages 8491–8499. ijcai.org, 2025.

[Esparza and Křetínský, 2014] Javier Esparza and Jan Křetínský. From LTL to deterministic automata: A safraless compositional approach. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 192–208. Springer, 2014.

[Esparza *et al.*, 2020] Javier Esparza, Jan Křetínský, and Salomon Sickert. A unified translation of linear temporal logic to ω-automata. *J. ACM*, 67(6):33:1–33:61, 2020.

[Giuri *et al.*, 2025] Mattia Giuri, Mathias Jackermeier, and Alessandro Abate. Zero-shot instruction following in rl via structured LTL representations. In *ICML Workshop on Programmatic Representations for Agent Learning*, 2025.

[Guo *et al.*, 2025] Zijian Guo, Ilker Isik, H. M. Sabbir Ahmad, and Wenchao Li. One subgoal at a time: Zero-shot generalization to arbitrary linear temporal logic requirements in multi-task reinforcement learning. In *NeurIPS*, 2025.

[Hahn *et al.*, 2019] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *TACAS (1)*, volume 11427 of *Lecture Notes in Computer Science*, pages 395–412. Springer, 2019.

[Hahn *et al.*, 2022] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. An impossibility result in automata-theoretic reinforcement learning. In *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings*, page 42–57, Berlin, Heidelberg, 2022. Springer-Verlag.

[Hasanbeig *et al.*, 2022] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. LCRL: certified policy synthesis via logically-constrained reinforcement learning. In *QEST*, volume 13479 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2022.

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

[Křetínský and Esparza, 2012] Jan Křetínský and Javier Esparza. Deterministic automata for the (f, g)-fragment of LTL. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2012.

[Křetínský *et al.*, 2018] Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018.

[Křetínský *et al.*, 2019] Jan Křetínský, Alexander Manta, and Tobias Meggendorfer. Semantic labelling and learning for parity game solving in LTL synthesis. In *ATVA*, volume 11781 of *Lecture Notes in Computer Science*, pages 404–422. Springer, 2019.

[Křetínský *et al.*, 2023] Jan Křetínský, Tobias Meggendorfer, Maximilian Prokop, and Sabine Rieder. Guessing winning policies in LTL synthesis by semantic learning. In *CAV (1)*, volume 13964 of *Lecture Notes in Computer Science*, pages 390–414. Springer, 2023.

[Křetínský *et al.*, 2025] Jan Křetínský, Tobias Meggendorfer, Maximilian Prokop, and Ashkan Zarkhah. Semml:

Enhancing automata-theoretic LTL synthesis with machine learning. In *TACAS (1)*, volume 15696 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 2025.

[León *et al.*, 2022] Borja G. León, Murray Shanahan, and Francesco Belardinelli. In a nutshell, the human asked for this: Latent goals for following temporal specifications. In *ICLR*. OpenReview.net, 2022.

[Li *et al.*, 2013] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. Ltl satisfiability checking revisited. In *Proceedings of the 2013 20th International Symposium on Temporal Representation and Reasoning*, TIME '13, page 91–98, USA, 2013. IEEE Computer Society.

[Liu *et al.*, 2022] Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. In *IJCAI*, pages 5502–5511. ijcai.org, 2022.

[Liu *et al.*, 2024] Jason Xinyu Liu, Ankit Shah, Eric Rosen, Mingxi Jia, George Konidaris, and Stefanie Tellex. Skill transfer for temporal task specification. In *ICRA*, pages 2535–2541. IEEE, 2024.

[Oh *et al.*, 2017] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

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

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

[Scarselli *et al.*, 2009] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

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

[Sickert and Esparza, 2020] Salomon Sickert and Javier Esparza. An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 831–844, New York, NY, USA, 2020. Association for Computing Machinery.

[Sickert *et al.*, 2016] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic büchi automata for linear temporal logic. In *CAV (2)*, volume 9780 of *Lecture Notes in Computer Science*, pages 312–332. Springer, 2016.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

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

[Voloshin *et al.*, 2023] Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 35137–35150. PMLR, 2023.

[Weinhuber *et al.*, 2026] Christoph Weinhuber, Giuseppe De Giacomo, Yong Li, Sven Schewe, and Qiyi Tang. Good-for-mdp state reduction for stochastic ltl planning. In *AAAI*. AAAI Press, 2026.

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