Zero-Shot Instruction Following in RL via Structured LTL Representations

Mathias Jackermeier[1] **Mattia Giuri**[1,2] **Jacques Cloete**[3] **Alessandro Abate**[1]
[1]Department of Computer Science, University of Oxford
[2]Qube Research and Technologies
[3]Oxford Robotics Institute, University of Oxford

Abstract

We study instruction following in multi-task reinforcement learning, where an agent must zero-shot execute novel tasks not seen during training. In this setting, linear temporal logic (LTL) has been adopted as a powerful framework for specifying structured, temporally extended tasks. While existing approaches successfully train generalist policies, they often struggle to effectively capture the rich logical and temporal structure inherent in LTL specifications. In this work, we address these concerns with a novel approach to learn *structured* task representations that facilitate training and generalisation. Our method conditions the policy on sequences of Boolean formulae constructed from a finite automaton of the task. We propose a hierarchical neural architecture to encode the logical structure of these formulae, and introduce an attention mechanism that enables the policy to reason about future subgoals. Experiments in a variety of complex domains demonstrate the strong generalisation capabilities and superior performance of our approach.

1 Introduction

Recent years have seen remarkable progress in training artificial intelligence agents to follow arbitrary instructions [35, 34, 38, 26]. A key consideration in this line of work is the *type* of instruction to provide to the agent. While a long line of research has investigated tasks expressed in natural language [37, 17, 21, 12], there recently has been increased interest in training agents to follow instructions specified in *formal* language [23, 50, 22]. Compared to natural language, formal specifications offer several advantages: they facilitate the training process by enabling the automatic construction of task-specific reward functions, they explicitly expose task structure via their compositional syntax, and they are grounded in well-defined semantics. This makes formal instruction languages especially appealing in safety-critical settings, where specifying unambiguous tasks is important [30].

One particular formal language that has emerged as an expressive formalism to specify tasks in reinforcement learning (RL) is *linear temporal logic* (LTL) [40]. LTL instructions are defined over a set of *atomic propositions*, corresponding to high-level events that can hold true or false at each state of the environment. These atomic propositions are combined using logical and temporal operators, which allow for the specification of complex, temporally extended behaviour in a compositional manner. Recent work has exploited the connection between LTL and corresponding automata structures (typically variants of Büchi automata [9]), which provide an automatic way to monitor task progress, to train generalist policies capable of executing arbitrary LTL instructions [42, 22, 18].

In this paper, we develop a novel approach to train generalist LTL-conditioned policies. Our method is designed to explicitly model the rich logical and temporal structure of LTL specifications, in order to yield *structured* task representations that accelerate learning and enable improved generalisation. We obtain these representations by translating transitions in the automaton into equivalent Boolean formulae, which provide succinct, compositional descriptions of the logical conditions required to

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

make progress towards a given task. We propose a hierarchical neural architecture to encode these formulae, and introduce an attention mechanism that allows the policy to reason about future subgoals, enabling more effective planning. Our main contributions are as follows:

- we propose representing LTL instructions as sequences of minimal Boolean formulae constructed from transitions in the corresponding Büchi automaton;
- we develop a structured representation learning approach to embed these sequences of formulae, based on a hierarchical neural architecture coupled with a temporal attention mechanism;
- we introduce a novel environment with complex logical structure, which allows us to systematically study the effectiveness of our approach across challenging LTL specifications;
- lastly, we conduct an extensive empirical evaluation demonstrating that our method achieves state-of-the-art results and significantly outperforms existing approaches across a wide range of tasks.

2 Related Work

Significant research effort has explored LTL as a task specification language for RL agents, with many works training agents to satisfy a *single*, fixed specification [44, 20, 11, 19, 7, 10, 48, 52, 29, 47]. In contrast, we aim to learn a general multi-task policy for zero-shot generalisation to unseen tasks.

Several approaches to this problem decompose LTL specifications into subtasks completed sequentially by a goal-conditioned policy [4, 30, 31, 42, 33, 18]. While this simplifies the learning process, such methods can exhibit *myopic* behaviour, since attending to a single subgoal at a time may not yield a globally optimal solution. Kuo et al. [27] instead compose RNNs mirroring the formula structure, though this requires learning a non-stationary policy, which is generally challenging [50].

Vaezipoor et al. [50] introduce LTL2Action, encoding the formula's syntax tree with a GNN and applying LTL progression [6] to continuously update the task representation. The primary drawback is that the policy must *learn* the semantics of temporal operators, and infinite-horizon tasks are not supported. Instead, we construct Büchi automata to explicitly capture the temporal structure of tasks.

Automata-based representations have previously been explored, but our work differs in how the task is represented and processed. Instead of learning a policy conditioned myopically on single atomic propositions [42], or on sequences of unstructured sets of assignments [22], we construct semantically meaningful Boolean formulae from automaton transitions, yielding a more explicit and structured representation of task dynamics. We furthermore incorporate information about future subgoals via a novel attention mechanism. Finally, recent work [14] explores scaling LTL-guided RL to parameterised predicates, which is orthogonal to our focus on structured task representations.

3 Background

Reinforcement Learning. We formally model reinforcement learning (RL) environments as Markov decision processes (MDPs). An MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discount factor, and ρ_0 is the initial state distribution. A (memoryless) policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps states to probability distributions over actions. The goal of RL is to find a policy π that maximises the expected discounted return $J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\tau = (s_0, a_0, r_0, s_1, \dots)$ is a trajectory generated by following π starting from $s_0 \sim \rho_0$, that is, $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$. The value function of π is defined as $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$, i.e. the expected discounted return of π starting from state s.

Linear Temporal Logic. We use linear temporal logic (LTL) [40] to specify tasks for RL agents. LTL formulae are defined over a set of atomic propositions (AP), which represent basic properties of the environment such as "object A is at location X" or "the agent is in the green region". LTL formulae are defined recursively as

$$\varphi ::= \top \mid \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{X}\,\varphi \mid \varphi\ \mathsf{U}\ \psi$$

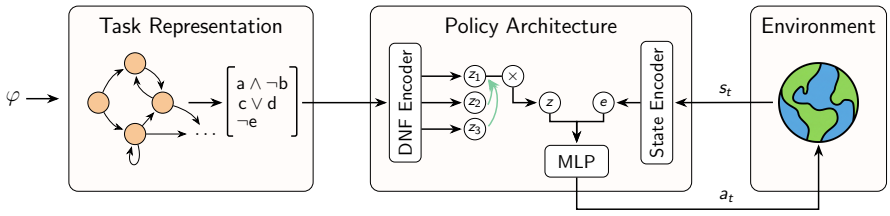


Figure 2: Overview of our method. *(Left)* Given an LTL instruction φ, we construct an LDBA and extract sequences of Boolean formulae representing the current automaton state. *(Right)* The policy is conditioned on a single formula sequence. It processes formulae with a hierarchical DNF encoder and a temporal attention mechanism to reason about future subgoals. The latent task representation z is concatenated with the encoded MDP state s_t and mapped to an action a_t via an actor MLP.

where $\top$ denotes true, $\mathsf{p} \in AP$ is an atomic proposition, φ and ψ are themselves LTL formulae, $\neg$ (negation) and $\wedge$ (conjunction) are standard Boolean connectives (from which others like $\vee$, $\rightarrow$, $\leftrightarrow$ can be derived), and X (next) and U (until) are temporal operators. From these, the common temporal operators $\mathsf{F}\,\varphi \equiv \top\ \mathsf{U}\ \varphi$ (eventually or finally) and $\mathsf{G}\,\varphi \equiv \neg\mathsf{F}\,\neg\varphi$ (always or globally) can be derived. Intuitively, $\varphi\ \mathsf{U}\ \psi$ is true if ψ becomes satisfied at some time step t and φ is true at all previous time steps. Accordingly, $\mathsf{F}\,\varphi$ simply states that φ has to be true *eventually*, whereas $\mathsf{G}\,\varphi$ requires φ to be true at every time step.

Formally, LTL semantics are defined over infinite traces of *truth assignments* $\sigma = \sigma_0\sigma_1\sigma_2\ldots$, where $\sigma_t \subseteq AP$ is the set of atomic propositions true at time t. We assume a (known) labelling function $L\colon \mathcal{S} \to 2^{AP}$, which maps MDP environment states $s \in \mathcal{S}$ to corresponding sets of true atomic propositions. A trajectory $\tau = (s_0, a_0, s_1, \ldots)$ satisfies an LTL formula φ, denoted $\tau \models \varphi$, if its trace $L(s_0)L(s_1)\ldots$ satisfies φ according to LTL semantics (see Appendix A). For example, the formula $\neg\mathsf{a}\ \mathsf{U}\ \mathsf{b}$ is satisfied by exactly the trajectories where eventually b is true at some time step t (i.e. $\mathsf{b} \in L(s_t)$), and a is false at all time steps before. LTL provides a formal and structured way to define complex tasks to RL agents, such as "eventually reach region A, and if you pick up an object on the way, you must also visit region B while avoiding region C". The problem of training a generalist LTL-guided policy can be formalised by defining a distribution $\mathcal{D}$ over LTL tasks. We then aim to compute the optimal task-conditioned policy

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

Büchi Automata. The semantics of LTL can equivalently be captured by Büchi automata [9], which provide a more explicit way to reason about tasks. We here focus on *limit-deterministic* Büchi automata (LDBAs) [49], which are defined as tuples $\mathcal{B} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{E}, \mathcal{F})$. $\mathcal{Q} = \mathcal{Q}_I \uplus \mathcal{Q}_A$ is a finite set of states partitioned into two subsets, $q_0 \in \mathcal{Q}$ 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. We require that $\mathcal{F} \subseteq \mathcal{Q}_A$ and $\delta(q, \cdot) \in \mathcal{Q}_A$ for all $q \in \mathcal{Q}_A$. The only way to transition from $\mathcal{Q}_I$ to $\mathcal{Q}_A$ is by taking an ε-transition $\varepsilon \in \mathcal{E}$ (a.k.a. *jump* transition), which does not consume any input. Given an input trace σ, a *run* of $\mathcal{B}$ is an infinite sequence of states in $\mathcal{Q}$ starting in q_0 and respecting the transition function δ. A trace is *accepted* by $\mathcal{B}$ if there exists a run that infinitely often visits accepting states. There exist standard algorithms, e.g. [49], to construct an LDBA $\mathcal{B}_\varphi$ from an LTL formula φ such that $\mathcal{B}_\varphi$ accepts exactly the traces satisfying φ.

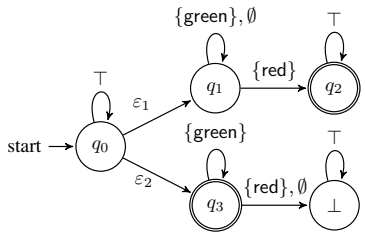


Figure 1: LDBA for the LTL formula $\mathsf{F}\,\mathsf{red} \vee (\mathsf{F}\,\mathsf{G}\,\mathsf{green})$.

Example 3.1. Figure 1 shows an LDBA for the formula $\mathsf{F}\,\mathsf{red} \vee (\mathsf{F}\,\mathsf{G}\,\mathsf{green})$. In q_0, the agent can either choose ε_1 to transition to q_1, from which it reaches the accepting state q_2 upon visiting an MDP state s with $\mathsf{red} \in L(s)$. By following ε_2, it instead has to stay in a green state forever. Note that we assume here that red and green cannot be true together.

Büchi automata are well-suited to represent LTL instructions, since they explicitly capture the memory required to execute a given task via their states. It is hence common to consider policies $\pi\colon \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$ conditioned not only on the current MDP state s, but also on the current LDBA state q [20, 19]. After each environment interaction, the automaton state is updated according to the transition function δ and the currently true propositions $L(s)$. Over the course of a trajectory, the policy can adjust its behaviour as the LDBA state changes. For example, in Figure 1 the policy would aim to make red true in q_1, but in q_3 instead try to stay in a green state.

4 Structured LTL Representations

In single-task RL, it is straightforward to condition the policy on LDBA states. Since the automaton remains fixed throughout training and evaluation, one can simply train separate state-specific sub-policies, and choose the appropriate one at test time (e.g. [23]). However, this approach is unfeasible in multi-task RL, since the policy must generalise to novel tasks (and therefore automata states) that it has never encountered during training. The key challenge is thus to find a suitable representation of LDBA states that (i) enables such generalisation, and (ii) is amenable to policy learning.

Recent work has explored various representations, such as conditioning on single atomic propositions [42], sequences of unstructured sets of assignments [22], or directly encoding the LTL formula syntax tree [50]. However, these approaches struggle to effectively capture the rich logical and temporal structure of tasks, particularly in environments where many propositions can hold simultaneously.

As illustrated in Figure 2, we instead propose representing LDBA states as *sequences of Boolean formulae*. This representation explicitly exposes the logical structure of tasks, enabling the policy to generalise compositionally. For example, it can leverage knowledge about achieving propositions a and b individually to also achieve $\mathsf{a} \wedge \mathsf{b}$. Furthermore, since the policy is conditioned on a *sequence* of such formulae, it can non-myopically reason about future subgoals, which may affect how it approaches the current step.

Intuitively, these sequences of Boolean formulae align with *accepting runs* in the LDBA, informing the policy of the high-level steps required to complete the task. We design a procedure to automatically construct such sequences from the LDBA (Section 4.1), and propose a specialised neural network architecture that leverages the rich information in these sequences to yield *structured* learned task representations (Section 4.2). Further details on the training procedure are given in Section 4.3. Finally, Section 4.4 describes the test-time execution, where the policy is conditioned on sequences extracted from the current task's LDBA and updates dynamically as the automaton state evolves.

4.1 Representing LDBA States with Boolean Formulae

Given an LDBA $\mathcal{B}_\varphi$ and state q, we aim to identify behaviour that will lead to satisfying the LTL instruction. Since φ is satisfied exactly when the agent visits accepting states infinitely often, we first enumerate possible ways to reach an *accepting cycle* in $\mathcal{B}_\varphi$, i.e. a cycle with at least one accepting state. This can be achieved with a simple depth-first search (DFS) starting from q (Algorithm 1 of Jackermeier and Abate [22]; see Appendix B for details).

The DFS yields a set of accepting runs $\rho_i = (q, q_1, \dots)$ that correspond to different ways of achieving φ. For each run ρ_i, we construct a sequence of Boolean formulae that captures the high-level goals the agent must complete in order to follow the run, and hence satisfy the LTL task. Specifically, in order to transition from q_i to q_{i+1}, the agent must achieve an assignment $\alpha_i \subseteq AP$ of atomic propositions that satisfies the transition condition $\delta(q_i, \alpha_i) = q_{i+1}$, while avoiding transitions to other states (excluding self-loops). We represent this with two Boolean formulae β_i^+ and β_i^- that satisfy

$$\begin{aligned} \forall \alpha \in \mathbb{A}.\, \alpha \models \beta_i^+ &\iff \delta(q_i, \alpha) = q_{i+1}, \\ \forall \alpha \in \mathbb{A}.\, \alpha \models \beta_i^- &\iff \delta(q_i, \alpha) \notin \{q_i, q_{i+1}\}, \end{aligned}$$

where $\mathbb{A} = \{L(s) : s \in \mathcal{S}\} \subseteq 2^{AP}$ is the set of possible assignments in the MDP (which we assume known). Intuitively, β_i^+ is only true for the assignments that transition from q_i to q_{i+1}, whereas β_i^- captures the assignments that must be avoided in order to avoid transitioning to other states. Boolean formulae can be much more succinct than explicitly enumerating sets of assignments, as we demonstrate with the following example.

Example 4.1. Consider an MDP with propositions $AP = \{a, b, c, d\}$ in which all combinations of propositions can hold true at the same time (i.e. $\mathbb{A} = 2^{AP}$). Assume we have an LDBA in which we can transition from state q_0 to q_1 via any of the assignments in the set

$$A = \big\{\{a\}, \{a, b\}, \{a, d\}, \{a, b, d\}\big\}.$$

The formula $\beta_0^+ \equiv a \wedge \neg c$ succinctly represents exactly this set of assignments.

Constructing Boolean formulae. We construct formulae β_i^+ and β_i^- for each step i in a run as follows: we first identify the relevant sets of assignments A_i^+ and A_i^- from the transition function δ, that is, $A_i^+ = \{\alpha \in \mathbb{A} : \delta(q_i, \alpha) = q_{i+1}\}$ and $A_i^- = \{\alpha \in \mathbb{A} : \delta(q_i, \alpha) \notin \{q_i, q_{i+1}\}\}$. We could then trivially construct formulae in *disjunctive normal form* (DNF) by exhaustively building a formula for each assignment. However, this representation is as verbose as simply listing the assignments themselves; instead, we aim to find small, semantically meaningful formulae that capture the assignments while enabling generalisation at test time.

Constructing minimal Boolean formulae is a well-studied problem in formal logic and the minimisation of digital circuits. We employ the widely used Quine-McCluskey algorithm [43, 36] to compute minimal DNF representations of β_i^+ and β_i^-. This algorithm takes as input a set of satisfying assignments S and a set of non-satisfying assignments N to produce a minimal (in terms of number of disjunctions and conjunctions) DNF formula that evaluates to true for assignments in S and to false for assignments in N. We set $S = A_i^+$ (or A_i^-) and $N = \mathbb{A} \setminus S$ to yield the formulae β_i^+ and β_i^-, respectively.

Note that an accepting run may contain an ε-transition representing a non-deterministic choice. These transitions do not consume input and hence require no Boolean formula; we instead represent them with a special token β_ε that we handle separately in our policy architecture (see Section 4.2).

4.2 Structured Representation Learning

Effectively conditioning a policy on the sequences derived in Section 4.1 presents two challenges: (i) the Boolean formulae possess hierarchical logical structure that unstructured embeddings fail to capture, and (ii) the sequences vary in length depending on the task's complexity. To address these challenges, we propose a hierarchical architecture to encode the structure of individual formulae and a temporal attention mechanism for the sequence of subgoals. This is illustrated in Figure 2 (right).

Formally, we denote by $\zeta = ((\beta_1^+, \beta_1^-), (\beta_2^+, \beta_2^-), \ldots)$ a sequence of Boolean formula pairs extracted from an accepting run. Our policy $\pi(a \mid s, \zeta)$ takes as input the current MDP state s and such a target sequence ζ. Note that while there may be multiple accepting runs for a given LDBA state, the policy is conditioned on a *single* formula sequence at any given time; we detail the run selection procedure in Section 4.4.

Embedding Boolean formulae. There are a number of straightforward choices to embed Boolean formulae, such as token-level sequence encoders (e.g. LSTMs or GRUs) or structure-aware encoders, e.g. applying a GNN to a syntax tree representation of the formula [16, 15]. Rather than using such general methods, we exploit two key features of the formulae encountered by our policy: (i) they only contain propositions from AP, and (ii) they are guaranteed to be in DNF by construction.

Specifically, a formula in DNF consists of a *disjunction* of one or more *clauses*, i.e. has the form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_{ij}$, where l_{ij} is a possibly negated atomic proposition. We process DNF formulae in a hierarchical fashion as follows. Our policy associates a trainable embedding $\boldsymbol{e}_a \in \mathbb{R}^d$ with each proposition $a \in AP$. Negated propositions are first processed with a learned linear projection layer, and literals in a clause are then aggregated into a single representation via a permutation-invariant set encoder (since the order of literals does not affect the semantics). In particular, we use a simple DeepSets encoder [54] consisting of a sum followed by a learned non-linear transformation. The resulting clause embeddings are aggregated again on the disjunction level by another DeepSets encoder, yielding the overall formula representation $\boldsymbol{z}$. Formally, we have

$$\boldsymbol{z} = f_\vee \left(\sum_{i=1}^{n} f_\wedge \left(\sum_{j=1}^{m_i} \Pi(\boldsymbol{e}_{a_{ij}}) \right) \right),$$

where $f_\vee$ and $f_\wedge$ are the non-linear transformations, and Π denotes the projection layer if l_{ij} is negated, and the identity function otherwise. Finally, note that this does not apply to the special β_ε token, for which we instead learn a separate embedding. We give more details on how we handle ε-transitions below.

Attending to future subgoals. In order to produce optimal behaviour, the policy may need to consider future subgoals rather than myopically focusing only on the first step of the sequence ζ. To illustrate, consider a simple sequential task where the agent needs to achieve a and then b. Depending on the underlying MDP, there may be different ways of making a true, some of which might render b infeasible (e.g. if the agent enters a room in order to achieve a that locks upon entry). At the same time, the first step of the sequence generally contains the most important information, since this is the goal the agent is currently trying to achieve. We model this intuition with a temporal attention mechanism that allows the policy to contextualise the representation of the current step by attending to future subgoals.

In particular, let $\boldsymbol{z}_i^+$ and $\boldsymbol{z}_i^-$ denote the embedding of β_i^+ and β_i^-, respectively. We first concatenate these embeddings to obtain the overall encoding of step i as $\boldsymbol{z}_i = [\boldsymbol{z}_i^+; \boldsymbol{z}_i^-]$. We use the first step $\boldsymbol{z}_1$ as the query to attend to the entire sequence using single-head scaled dot-product attention [51], which computes the output

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

where $\boldsymbol{q} = \boldsymbol{z}_1 W^Q$ is the query vector, and $K, V \in \mathbb{R}^{T \times d_k}$ are the key and value matrices obtained by projecting the sequence embeddings $\boldsymbol{z}_{1:T}$ using W^K and W^V, respectively. W^O is the output projection matrix. Intuitively, $\boldsymbol{h}$ captures how future formulae in the sequence modulate the current goal $\boldsymbol{z}_1$. We obtain the final contextualised goal representation as $\tilde{\boldsymbol{z}}_1 = \text{LayerNorm}(\boldsymbol{z}_1 + \boldsymbol{h})$ [5]. Since pure attention does not preserve sequential information, we employ ALiBi [41] to bias the attention towards nearby steps. Specifically, before applying the softmax, we subtract a linear penalty $m \cdot (i-1)$ from the unnormalised attention score for step i (where $i \geq 2$), with m being a fixed hyperparameter. This augments the embeddings with positional information and captures the intuition that steps further in the future tend to have less influence on the immediate goal.

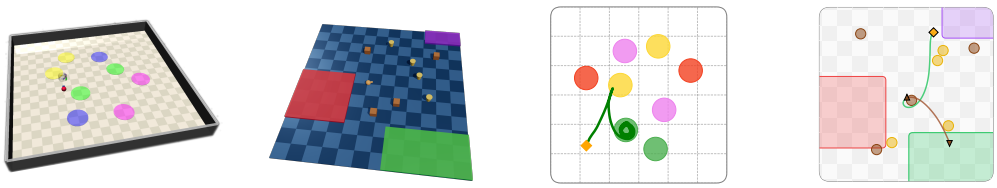
Finally, note that formula sequences constructed from accepting runs are technically infinite. We hence truncate sequences after repeating the accepting cycle a finite number of times [22]. Crucially, the policy is still always conditioned on a complete formula sequence leading to an accepting cycle.

Full policy architecture. As depicted in Figure 2 (right), our policy not only processes the formula sequence, but also encodes the current MDP state via a *state encoder*. This is either a multilayer perceptron (MLP) or convolutional neural network (CNN), depending on the environment. The contextualised formula sequence embedding $\tilde{\boldsymbol{z}}_1$ is concatenated with the encoded MDP state and fed into a final *actor* MLP, which produces a distribution over actions.

We handle ε-transitions in the LDBA by augmenting the action space of the policy with designated ε-actions, as commonly done [20, 52]. These do not perform an action in the MDP, but instead simply update the current LDBA state to the target of the corresponding ε-transition, hence updating the product MDP state (s, q). Our policy produces a mixture distribution, where it takes an ε-action with a learned parameter p and otherwise samples from its distribution over MDP actions.

4.3 Training Procedure

We optimize the parameters of our model end-to-end via goal-conditioned RL [34]. That is, we sample a random Boolean formula sequence ζ at the beginning of each training episode, which sometimes contain the β_ε-token. We then execute a rollout with the policy conditioned on ζ, assigning a positive reward of $+1$ to an episode if the agent successfully sequentially satisfies all formulae β_i^+. If the agent at any step i instead satisfies the currently active β_i^-, we assign a negative reward of -1. This reward structure encourages the policy to satisfy formula sequences that lead to accepting cycles, and hence approximates Equation (1). We use *proximal policy optimization* (PPO) [46] to jointly optimise the policy and learn a value function V^π. To further improve training convergence in practice, we follow Jackermeier and Abate [22] and use *curriculum learning* to gradually increase the complexity of formula sequences encountered during training (see Appendix C.3 for details).



(a) ZoneEnv-NM (b) WarehouseEnv (c) F yellow ∧ F G green (d) Deliver crate to B (2)

Figure 3: Environment visualisations (left) and example StructLTL trajectories (right). In ZoneEnv-NM, the agent navigates between coloured zones. In Warehouse, the agent moves crates and vases between different regions. Both environments provide high-dimensional lidar observations and continuous movement actions. The orange diamond indicates the starting position of the agent, and triangles pointing upwards and downwards indicate picking up and dropping an object, respectively.

4.4 Test-time Policy Execution

At test time, we are given a new LTL instruction φ and build a corresponding LDBA $\mathcal{B}_\varphi$. We then perform the DFS described in Section 4.1 to extract a set of Boolean formula sequences $\{\zeta_1, \dots, \zeta_n\}$ from the initial state q_0. We use the trained *value function* V^π to select the formula sequence that the policy is approximately most likely able to achieve from the current MDP state s, i.e. we compute $\zeta^* = \arg\max_{\zeta_i} V^\pi(s, \zeta)$. Intuitively, ζ^* corresponds to the path through the automaton with the highest likelihood of success. The value function, which is a by-product of the training algorithm, naturally estimates (a lower bound of) this likelihood and is hence an appropriate choice for selecting a formula sequence. We then execute the trained policy π conditioned on ζ^*, i.e. continuously sample actions $a_t \sim \pi(\cdot \mid s_t, \zeta^*)$, until the LDBA state changes, upon which we recompute and update the formula sequence used to condition the policy.

5 Experiments

We evaluate our method across diverse environments, including high-dimensional domains with continuous action spaces. We aim to answer the following research questions:[1] **(Q1)** Can our approach effectively zero-shot generalise to unseen LTL instructions, particularly those requiring compositional reasoning? **(Q2)** Can our method leverage global task information for non-myopic execution? **(Q3)** How does our method compare to state-of-the-art baselines in terms of sample efficiency and asymptotic performance?

Environments. We conduct experiments in *ZoneEnv* [50] (Figure 3a), a well-established high-dimensional robotic navigation environment, in which the agent observes differently coloured zones via a lidar sensor. The colours correspond to the atomic propositions, which hold true when the agent is located in a corresponding zone. To increase the complexity of the environment, we introduce state-modifying behaviour requiring non-myopic reasoning (denoted as *ZoneEnv-NM*): when the agent first enters a purple zone, all green zones disappear for the rest of the episode. The agent must learn to reason about this behaviour in order to successfully complete complex tasks.

We furthermore introduce the *WarehouseEnv* environment (Figure 3b), which admits complex logical specifications. It consists of an agent navigating fixed regions while interacting with *crates* and *vases* that are randomly scattered across the environment. The environment features a hybrid action space: the agent can both apply a continuous movement action and choose from a set of discrete actions to pick up and drop objects. Notably, WarehouseEnv supports complex combinations of atomic propositions holding true at the same time. For example, if the agent is in region A while carrying both a vase and crate, all three of these propositions hold true. Completing tasks in the warehouse environment thus requires the agent to reason compositionally about interactions between propositions. See Appendix C.2 for further details on the environments.

Baselines. We compare our approach, named StructLTL, to state-of-the-art approaches for learning generalist LTL-conditioned policies. LTL2Action [50] directly encodes the syntax tree of a given

[1] Our code is available at github.com/mathiasj33/structltl.

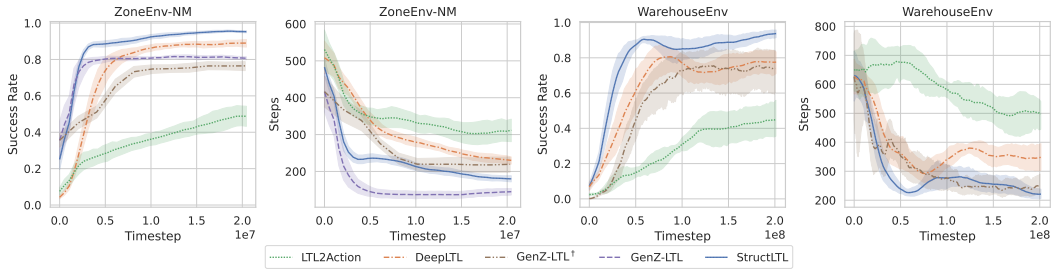


Figure 4: Evaluation curves on finite-horizon tasks over training. We report averages over the entire set of finite-horizon evaluation tasks, computed over 50 episodes per task. Shaded areas indicate 95% confidence intervals over 10 random seeds.

Table 1: Evaluation results of trained policies on finite- and infinite-horizon tasks. Results are averaged over 10 seeds and 512 episodes per seed, with 95% bootstrapped confidence intervals. — indicates not applicable. See Appendix C.5 for details on the task specifications.

		Finite Horizon						**Infinite Horizon**			
		Success Rate (↑)						Average Visits (↑)			
	φ	LTL2Action	DeepLTL	GenZ-LTL†	GenZ-LTL	StructLTL	ψ	DeepLTL	GenZ-LTL†	GenZ-LTL	StructLTL
ZoneEnv-NM	φ_1	$0.18^{+0.10}_{-0.07}$	$0.87^{+0.03}_{-0.03}$	$0.81^{+0.04}_{-0.04}$	$0.92^{+0.01}_{-0.01}$	$\mathbf{0.93}^{+0.01}_{-0.01}$	ψ_1	$2.16^{+00.37}_{-00.36}$	$1.96^{+0.47}_{-0.47}$	$\mathbf{4.44}^{+00.25}_{-00.22}$	$3.08^{+00.16}_{-00.15}$
	φ_2	$0.59^{+0.15}_{-0.15}$	$0.96^{+0.01}_{-0.01}$	$0.99^{+0.01}_{-0.01}$	$0.97^{+0.03}_{-0.04}$	$\mathbf{0.99}^{+0.01}_{-0.01}$	ψ_2	$2.43^{+00.84}_{-00.85}$	$2.68^{+0.52}_{-0.51}$	$4.10^{+00.38}_{-00.36}$	$\mathbf{4.55}^{+00.48}_{-00.42}$
	φ_3	$0.51^{+0.16}_{-0.17}$	$0.85^{+0.04}_{-0.04}$	$0.84^{+0.03}_{-0.04}$	$0.93^{+0.01}_{-0.01}$	$\mathbf{0.94}^{+0.01}_{-0.01}$	ψ_3	$2.31^{+00.41}_{-00.37}$	$1.71^{+0.20}_{-0.21}$	$2.78^{+00.10}_{-00.09}$	$\mathbf{3.19}^{+00.41}_{-00.49}$
	φ_4	$0.75^{+0.11}_{-0.13}$	$0.94^{+0.02}_{-0.02}$	$0.93^{+0.02}_{-0.02}$	$\mathbf{0.99}^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$	ψ_4	$563.08^{+55.74}_{-65.50}$	$42.97^{+9.50}_{-9.26}$	$411.57^{+101.14}_{-96.55}$	$\mathbf{646.26}^{+55.04}_{-47.89}$
	φ_5	$0.91^{+0.03}_{-0.03}$	$0.96^{+0.01}_{-0.02}$	$0.98^{+0.01}_{-0.01}$	$\mathbf{1.00}^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$	ψ_5	$356.14^{+42.70}_{-46.19}$	$30.82^{+7.28}_{-7.22}$	$332.52^{+78.00}_{-77.14}$	$\mathbf{465.48}^{+44.85}_{-40.86}$
	φ_6	$0.28^{+0.13}_{-0.13}$	$0.88^{+0.02}_{-0.02}$	$0.52^{+0.09}_{-0.10}$	$0.49^{+0.02}_{-0.02}$	$\mathbf{0.94}^{+0.01}_{-0.02}$	ψ_6	$546.22^{+103.85}_{-104.59}$	$32.68^{+6.30}_{-6.00}$	$407.31^{+97.10}_{-90.89}$	$\mathbf{651.15}^{+73.60}_{-81.99}$
	φ_7	$0.35^{+0.10}_{-0.09}$	$0.89^{+0.02}_{-0.03}$	$0.59^{+0.10}_{-0.11}$	$0.65^{+0.01}_{-0.01}$	$\mathbf{0.94}^{+0.01}_{-0.02}$	ψ_7	$527.50^{+111.03}_{-115.67}$	$28.99^{+5.09}_{-4.83}$	$374.16^{+84.21}_{-78.07}$	$\mathbf{672.39}^{+72.40}_{-82.47}$
	φ_8	$0.35^{+0.15}_{-0.13}$	$0.78^{+0.05}_{-0.05}$	$0.50^{+0.09}_{-0.09}$	$0.51^{+0.02}_{-0.02}$	$\mathbf{0.92}^{+0.02}_{-0.02}$	ψ_8	$361.99^{+73.37}_{-77.65}$	$35.78^{+6.64}_{-5.56}$	$191.42^{+64.64}_{-60.49}$	$\mathbf{491.31}^{+83.62}_{-91.77}$
WarehouseEnv	φ_9	$0.98^{+0.01}_{-0.02}$	$0.98^{+0.01}_{-0.01}$	$\mathbf{0.99}^{+0.01}_{-0.01}$	—	$0.99^{+0.01}_{-0.01}$	ψ_9	$2.52^{+00.19}_{-00.19}$	$2.45^{+00.22}_{-00.22}$	—	$\mathbf{2.78}^{+00.07}_{-00.07}$
	φ_{10}	$0.56^{+0.25}_{-0.29}$	$0.95^{+0.01}_{-0.01}$	$0.84^{+0.10}_{-0.19}$	—	$\mathbf{0.95}^{+0.01}_{-0.01}$	ψ_{10}	$13.84^{+03.15}_{-02.73}$	$13.21^{+06.05}_{-04.65}$	—	$\mathbf{15.92}^{+07.55}_{-06.48}$
	φ_{11}	$0.46^{+0.26}_{-0.25}$	$0.99^{+0.01}_{-0.02}$	$0.91^{+0.08}_{-0.13}$	—	$\mathbf{0.99}^{+0.00}_{-0.00}$	ψ_{11}	$2.32^{+00.32}_{-00.37}$	$2.27^{+00.63}_{-00.82}$	—	$\mathbf{3.19}^{+00.17}_{-00.17}$
	φ_{12}	$0.36^{+0.23}_{-0.22}$	$0.88^{+0.03}_{-0.04}$	$\mathbf{0.93}^{+0.04}_{-0.05}$	—	$0.80^{+0.08}_{-0.09}$	ψ_{12}	$2.07^{+00.66}_{-00.70}$	$2.35^{+00.69}_{-00.85}$	—	$\mathbf{3.93}^{+00.39}_{-00.35}$
	φ_{13}	$0.42^{+0.32}_{-0.28}$	$0.91^{+0.05}_{-0.06}$	$0.80^{+0.20}_{-0.30}$	—	$\mathbf{0.99}^{+0.00}_{-0.00}$	ψ_{13}	$823.49^{+81.49}_{-128.61}$	$131.88^{+72.50}_{-52.17}$	—	$\mathbf{880.60}^{+30.81}_{-35.05}$
	φ_{14}	$0.62^{+0.28}_{-0.32}$	$0.58^{+0.23}_{-0.24}$	$0.78^{+0.20}_{-0.29}$	—	$\mathbf{0.97}^{+0.02}_{-0.03}$	ψ_{14}	$433.90^{+146.89}_{-152.40}$	$114.73^{+49.09}_{-48.11}$	—	$\mathbf{656.79}^{+127.06}_{+163.11}$
	φ_{15}	$0.41^{+0.28}_{-0.31}$	$0.67^{+0.17}_{-0.19}$	$0.06^{+0.05}_{-0.04}$	—	$\mathbf{0.95}^{+0.03}_{-0.06}$	ψ_{15}	$650.87^{+187.21}_{-196.31}$	$10.02^{+08.20}_{-06.46}$	—	$\mathbf{682.62}^{+122.94}_{-133.97}$
	φ_{16}	$0.12^{+0.24}_{-0.12}$	$0.57^{+0.15}_{-0.21}$	$0.74^{+0.17}_{-0.21}$	—	$\mathbf{0.84}^{+0.10}_{-0.12}$	ψ_{16}	$219.22^{+188.46}_{-162.27}$	$13.98^{+13.27}_{-09.35}$	—	$\mathbf{351.43}^{+115.60}_{-123.51}$
	φ_{17}	$0.15^{+0.23}_{-0.14}$	$0.66^{+0.19}_{-0.21}$	$0.71^{+0.20}_{-0.24}$	—	$\mathbf{0.96}^{+0.02}_{-0.10}$	ψ_{17}	$\mathbf{783.72}^{+105.80}_{-122.94}$	$212.38^{+172.09}_{-140.50}$	—	$577.07^{+170.65}_{-172.36}$
	φ_{18}	$0.06^{+0.08}_{-0.06}$	$0.52^{+0.19}_{-0.19}$	$0.74^{+0.18}_{-0.22}$	—	$\mathbf{0.88}^{+0.10}_{-0.18}$	ψ_{18}	$575.43^{+174.13}_{-177.15}$	$113.44^{+139.29}_{-101.65}$	—	$\mathbf{857.86}^{+25.25}_{-31.28}$

specification using a GNN. We also compare to DeepLTL [22], the state-of-the-art non-myopic approach based on Büchi automata. Finally, we include the state-of-the-art myopic method GenZ-LTL [18], which simplifies the learning problem with hand-designed observation reduction and fusion mechanisms. Since these are not applicable in the warehouse environment (see Appendix C.4 for further discussion), we also compare to a one-hot subgoal variant of GenZ-LTL, denoted as GenZ-LTL†. Hyperparameters and further details are given in Appendix C.4.

Tasks. We consider diverse tasks of varying complexity to evaluate the trained policies. We differentiate between *finite-horizon* (i.e. co-safety) tasks, which can be completed in a finite number of steps, and *infinite-horizon* tasks, which specify recurrent behavior that the agent must execute indefinitely. Our tasks span a wide range of environment-specific behaviour, such as reaching and avoiding different zones in ZoneEnv-NM, or transporting objects between regions in the Warehouse environment. They furthermore make use of complex LTL constructs, such as implications to specify *response* tasks and nested temporal operators. As an example, task φ_{15} of our evaluation is given by

$$(\mathsf{crate} \rightarrow (\mathsf{crate}\ \mathsf{U}\ \mathsf{region_b}))\ \mathsf{U}\ (\mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\ \mathsf{U}\ (\neg\mathsf{crate} \wedge \mathsf{region_b}))), \quad (2)$$

which can be read as "Pick up a crate and place it in region B. After you have picked up the crate, you must not put it down again, until you have reached the region." We provide a full list of evaluation specifications in Appendix C.5.

Evaluation protocol. DeepLTL and StructLTL are trained with the same generic training curriculum (see Appendix C.3 for further details). LTL2Action originally does not employ curriculum training, but we found this to be beneficial and thus use an adapted training curriculum for a fair comparison.

Table 2: Average steps to successful completion (↓) on finite-horizon tasks, with 95% bootstrap confidence intervals; protocol as in Table 1.

ZoneEnv-NM

φ	LTL2Action	DeepLTL	GenZ-LTL†	GenZ-LTL	StructLTL
φ_1	441.6^{+47}_{-54}	275.7^{+25}_{-24}	281.0^{+29}_{-32}	$\mathbf{178.6}^{+09}_{-10}$	222.0^{+10}_{-10}
φ_2	421.2^{+40}_{-44}	257.6^{+22}_{-17}	252.9^{+26}_{-26}	$\mathbf{188.2}^{+30}_{-26}$	193.5^{+12}_{-11}
φ_3	334.6^{+75}_{-74}	201.9^{+17}_{-16}	183.2^{+17}_{-16}	$\mathbf{125.6}^{+05}_{-06}$	156.8^{+10}_{-10}
φ_4	289.9^{+58}_{-52}	186.7^{+18}_{-16}	175.3^{+17}_{-18}	$\mathbf{121.6}^{+06}_{-07}$	139.9^{+07}_{-07}
φ_5	148.0^{+20}_{-20}	135.5^{+12}_{-12}	126.2^{+10}_{-11}	$\mathbf{83.9}^{+04}_{-04}$	102.8^{+05}_{-06}
φ_6	450.2^{+58}_{-59}	291.0^{+15}_{-14}	280.6^{+16}_{-17}	$\mathbf{180.5}^{+11}_{-13}$	238.0^{+14}_{-13}
φ_7	192.0^{+45}_{-40}	179.6^{+15}_{-15}	163.5^{+16}_{-16}	$\mathbf{111.9}^{+07}_{-08}$	125.6^{+05}_{-05}
φ_8	566.3^{+38}_{-36}	333.5^{+17}_{-18}	338.1^{+26}_{-29}	$\mathbf{227.0}^{+16}_{-18}$	276.1^{+17}_{-16}

WarehouseEnv

φ	LTL2Action	DeepLTL	GenZ-LTL†	StructLTL
φ_9	226.3^{+013}_{-013}	$\mathbf{196.5}^{+05}_{-05}$	221.4^{+09}_{-08}	202.1^{+08}_{-07}
φ_{10}	169.7^{+010}_{-008}	163.4^{+18}_{-16}	157.2^{+11}_{-09}	$\mathbf{143.7}^{+07}_{-06}$
φ_{11}	346.2^{+093}_{-097}	$\mathbf{160.2}^{+13}_{-10}$	179.2^{+17}_{-17}	173.2^{+25}_{-19}
φ_{12}	387.2^{+046}_{-040}	353.1^{+35}_{-27}	$\mathbf{349.8}^{+17}_{-18}$	371.2^{+21}_{-19}
φ_{13}	331.3^{+136}_{-120}	185.8^{+31}_{-28}	201.4^{+89}_{-65}	$\mathbf{126.8}^{+03}_{-03}$
φ_{14}	304.3^{+102}_{-079}	323.3^{+81}_{-69}	202.0^{+08}_{-09}	$\mathbf{193.3}^{+15}_{-10}$
φ_{15}	220.8^{+059}_{-053}	151.3^{+20}_{-18}	210.0^{+61}_{-62}	$\mathbf{139.5}^{+06}_{-06}$
φ_{16}	573.2^{+115}_{-075}	430.5^{+55}_{-52}	338.5^{+30}_{-25}	$\mathbf{329.3}^{+35}_{-30}$
φ_{17}	378.2^{+088}_{-064}	211.4^{+54}_{-41}	169.0^{+22}_{-15}	$\mathbf{151.1}^{+05}_{-04}$
φ_{18}	613.4^{+095}_{-125}	449.4^{+53}_{-54}	339.3^{+41}_{-33}	$\mathbf{300.0}^{+23}_{-17}$

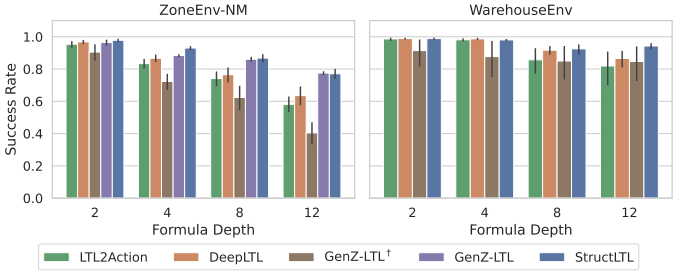


Figure 5: Generalisation to reach-tasks of increasing depth.

	DeepLTL	GenZ-LTL†	StructLTL
φ_{19}	$0.78^{+0.13}_{-0.14}$	$0.77^{+0.18}_{-0.23}$	$\mathbf{0.97}^{+0.02}_{-0.04}$
φ_{20}	$0.70^{+0.18}_{-0.21}$	$0.75^{+0.17}_{-0.22}$	$\mathbf{0.96}^{+0.02}_{-0.03}$
φ_{21}	$0.70^{+0.21}_{-0.25}$	$0.57^{+0.22}_{-0.24}$	$\mathbf{0.95}^{+0.02}_{-0.03}$
φ_{22}	$0.04^{+0.06}_{-0.04}$	$0.59^{+0.25}_{-0.27}$	$\mathbf{0.96}^{+0.03}_{-0.05}$
φ_{23}	$0.11^{+0.15}_{-0.11}$	$0.65^{+0.20}_{-0.23}$	$\mathbf{0.89}^{+0.09}_{-0.16}$
φ_{24}	$0.12^{+0.17}_{-0.11}$	$0.45^{+0.23}_{-0.22}$	$\mathbf{0.77}^{+0.16}_{-0.21}$

Table 3: Compositional generalisation results over 10 seeds and 512 episodes per seed, with 95% bootstrapped confidence intervals.

We train using PPO [46], with detailed hyperparameters in Appendix C.4. GenZ-LTL instead uses a safety-aware PPO procedure [53] and is trained by sampling feasible reach-avoid subgoals. We follow standard practice [42, 22, 18] and report success rates and number of steps until successful completion for finite-horizon tasks, and the number of successfully completed accepting cycles over a fixed evaluation horizon of 1000 steps for infinite-horizon tasks. Note that LTL2Action is not applicable to infinite-horizon tasks due to its reliance on LTL progression, which never yields positive reward signals for non-co-safe formulae. All results are averaged over 10 random seeds and 512 episodes per seed, and we provide 95% stratified bootstrap confidence intervals [2].

5.1 Results

Figure 4 shows evaluation curves averaged over finite-horizon tasks over training, and Table 1 lists results of the trained policies on both finite- and infinite-horizon specifications. Table 2 reports the number of steps until successful completion of finite-horizon tasks, and Figure 6 shows violation rates over training. We visualise trajectories produced by StructLTL in Figures 3c and 3d and Appendix F.

StructLTL consistently achieves high success rates in finite-horizon tasks, in most cases exceeding 90%. This is true not only for the simpler specifications in ZoneEnv-NM, but also for complex tasks requiring compositional reasoning in the Warehouse environment (**Q1**). In comparison to the baselines, we overall achieve significantly higher performance, particularly on tasks requiring non-myopic reasoning ($\varphi_6 - \varphi_8$) (**Q2**) and involving complex combinations of propositions ($\varphi_{14} - \varphi_{18}$). Violation rates are also generally lower than for the baselines. These results highlight the benefits of our structured representations: whereas our approach represents complex combinations of atomic propositions via succinct Boolean formulae, DeepLTL and GenZ-LTL struggle to handle the combinatorial explosion of assignments. LTL2Action underperforms even on simpler tasks, consistent with previously reported results [22, 18] (**Q3**).

Notably, StructLTL achieves similar success rates to GenZ-LTL in ZoneEnv-NM, even when the latter employs a hand-crafted, environment-specific observation reduction function. Without this observation reduction its performance notably degrades (GenZ-LTL†), and our method thus performs significantly better. However, we observe that the reduced observation space — where applicable — allows GenZ-LTL to learn more efficient policies, requiring fewer steps until task completion. In WarehouseEnv, where GenZ-LTL's observation reduction is not applicable, StructLTL generally

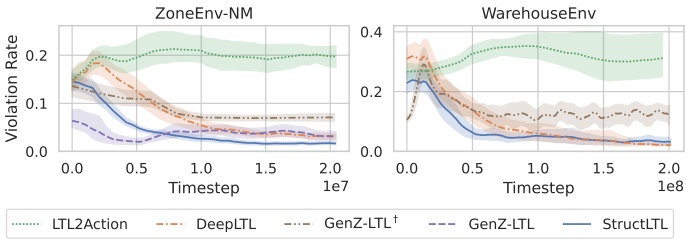


Figure 6: Violation rates over training on tasks that can be violated per environment (50 episodes each). Shading shows 95% confidence intervals across seeds.

achieves significantly higher success rates while simultaneously requiring fewer steps. The results on infinite-horizon tasks paint a similar picture: our approach consistently completes more accepting cycles in the LDBA for almost all formulae, suggesting that it is both better and more efficient at achieving challenging infinite-horizon specifications.

Generalisation study. We investigate the generalisation ability of our approach to long Boolean formulae sequences. During training, we sample sequences of maximum length 2. Figure 5 shows success rates obtained on randomly sampled nested LTL formulae $\mathsf{F}(\psi_1 \wedge \mathsf{F}(\psi_2 \wedge \ldots))$ of depth up to 12. While longer formulae sequences are naturally more challenging, we observe that StructLTL achieves high success rates on these out-of-distribution tasks and performs better than the baselines.

We furthermore conduct a compositional generalisation experiment: we train baseline methods and StructLTL in WarehouseEnv with a maximum conjunction length of 2, and test on tasks involving unseen conjunctions of length 3. Task details are provided in Appendix C.5. The results in Table 3 show that StructLTL successfully generalises to tasks involving unseen Boolean formulae.

Further results and ablations. In Appendix D.1 we further validate the non-myopic reasoning capabilities of our approach. Furthermore, we conduct ablation studies in Appendix D.2 to analyse the impact of the hierarchical DNF encoder (comparing to GNN and RNN variants) and the temporal attention mechanism. We find that both components significantly contribute to StructLTL's performance. We also ablate the impact of curriculum learning (Appendix D.3), demonstrating that it leads to improved sample efficiency and performance. At the same time, the results show that our method is also able to learn successfully with no curriculum at all. We analyse the computational complexity and practical runtime of our method in Appendix E.

6 Conclusion, Limitations & Future Work

We presented StructLTL, a novel approach for learning generalist policies to follow LTL instructions based on structured task representations. Our method exploits the structure of Büchi automata constructed from a given specification, extracting sequences of Boolean formulae that succinctly represent different ways of achieving the task. These formulae are encoded with a specialised hierarchical neural network architecture, and we furthermore introduce a temporal attention mechanism to enable non-myopic reasoning about future subgoals. In contrast to previous methods, our structured representations effectively capture interactions between atomic propositions, thereby enabling compositional generalisation. We demonstrated that our approach can effectively zero-shot generalise to unseen LTL instructions and outperforms state-of-the-art methods across complex domains and tasks.

A limitation of our method is that — like previous approaches — we rely on ground-truth access to the labelling function L (which maps from observations to atomic propositions). We are excited to explore ideas such as jointly learning the labelling function from visual or sensory input, or leveraging pre-trained foundation models as high-level event detectors in the future, as investigated in recent work [32, 39]. Another interesting direction for future work is to incorporate techniques such as hindsight experience replay [3, 52] to improve the sample efficiency and performance of our method. We believe that future advances in these areas could directly lead to real-world applications, such as in autonomous robotic domains.

Acknowledgments

This work was supported by the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI (No. EP/Y028872/1). MJ and JC are funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1).

References

[1] Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Kretínský, Maximilian Prokop, and Christoph Weinhuber. Semantically labelled automata for multi-task reinforcement learning with LTL instructions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3–11, 2026.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pages 29304–29320, 2021.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[4] 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*, pages 307–317, 2021.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv*, 2016.

[6] 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.

[7] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10349–10355, 2020. doi: 10.1109/ICRA40945.2020.9196796.

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

[9] 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*, pages 1–11. Elsevier, January 1966. doi: 10.1016/S0049-237X(09)70564-6.

[10] Mingyu Cai, Shaoping Xiao, Baoluo Li, Zhiliang Li, and Zhen Kan. Reinforcement Learning Based Temporal Logic Control with Maximum Probabilistic Satisfaction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 806–812, 2021. doi: 10.1109/ICRA48506.2021.9561903.

[11] 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*, pages 6065–6073, 2019. doi: 10.24963/IJCAI.2019/840.

[12] Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*, pages 3676–3713. PMLR, July 2023.

[13] 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*, pages 103–111. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-4012.

[14] Jacques Cloete, Mathias Jackermeier, Ioannis Havoutis, and Alessandro Abate. PlatoLTL: Learning to generalize across symbols in LTL instructions for multi-task RL. *arXiv*, January 2026.

[15] Maxwell Crouse, Ibrahim Abdelaziz, Cristina Cornelio, Veronika Thost, Lingfei Wu, Kenneth Forbus, and Achille Fokoue. Improving Graph Neural Network Representations of Logical Formulae with Subgraph Pooling, 2020.

[16] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can Neural Networks Understand Logical Entailment? In *International Conference on Learning Representations*, 2018.

[17] 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, pages 2385–2391. AAAI Press, 2019. ISBN 978-0-9992411-4-1.

[18] Zijian Guo, İlker Işık, H. M. Sabbir Ahmad, and Wenchao Li. One Subgoal at a Time: Zero-Shot Generalization to Arbitrary Linear Temporal Logic Requirements in Multi-Task Reinforcement Learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, October 2025.

[19] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412, 2019. ISBN 978-3-030-17462-0. doi: 10.1007/978-3-030-17462-0_27.

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

[21] Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human Instruction-Following with Deep Reinforcement Learning via Transfer-Learning from Text. In *arXiv*, May 2020. doi: 10.48550/arXiv.2005.09382.

[22] Mathias Jackermeier and Alessandro Abate. DeepLTL: Learning to efficiently satisfy complex LTL specifications for multi-task RL. In *International Conference on Learning Representations (ICLR)*, 2025.

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

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

[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[26] Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C. Machado, and Pierluca D'Oro. Maestro-Motif: Skill Design from Artificial Intelligence Feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

[27] 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)*, pages 5604–5610, October 2020. doi: 10.1109/IROS45743.2020.9341325.

[28] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to Your Favourite Deterministic Automaton. In *Computer Aided Verification (CAV)*, volume 10981, pages 567–577, 2018.

[29] 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.

[30] 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.

[31] 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.

[32] Andrew C. Li, Toryn Q. Klassen, Andrew Wang, Parand A. Alamdari, and Sheila A. McIlraith. Ground-compose-reinforce: Grounding language in agentic behaviours using limited data. In *NeurIPS*, 2025.

[33] 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)*, pages 2535–2541, 2024. doi: 10.1109/ICRA57147.2024.10611432.

[34] 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*, pages 5502–5511, July 2022. ISBN 978-1-956792-00-3. doi: 10.24963/ijcai.2022/770.

[35] 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*, pages 6309–6317, 2019. doi: 10.24963/IJCAI.2019/880.

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

[37] 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*, pages 2661–2670. PMLR, July 2017.

[38] Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking Agentic LLM and VLM Reasoning On Games. In *The Thirteenth International Conference on Learning Representations*, 2025.

[39] Roko Parac, Christos Ziakas, Leo Ardon, and Alessandra Russo. Rm-vlm: From natural-language task descriptions to vision-grounded reward machines, 2026. URL https://arxiv.org/abs/2611.08308.

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

[41] Ofir Press, Noah Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*, October 2021.

[42] 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.

[43] W. V. Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.

[44] 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*, pages 1091–1096, December 2014. doi: 10.1109/CDC.2014.7039527.

[45] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[46] 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.

[47] Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A. Seshia. LTL-Constrained Policy Optimization with Cycle Experience Replay. *Transactions on Machine Learning Research*, 2025.

[48] 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*, pages 4180–4189, 2023. doi: 10.24963/IJCAI.2023/465.

[49] 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, pages 312–332, 2016. ISBN 978-3-319-41540-6. doi: 10.1007/978-3-319-41540-6_17.

[50] 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*, pages 10497–10508. PMLR, July 2021.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Annual Conference on Neural Information Processing Systems 2017*, pages 5998–6008, 2017.

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

[53] Dongjie Yu, Haitong Ma, Shengbo Li, and Jianyu Chen. Reachability constrained reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 25636–25655, Jul 2022.

[54] 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 Semantics of LTL

The satisfaction semantics of LTL are specified by the satisfaction relation $\sigma \models \varphi$, which is recursively defined as follows [40]:

$$
\begin{aligned}
&\sigma \models \top && \\
&\sigma \models \mathsf{a} && \text{iff } \mathsf{a} \in \sigma_0 \\
&\sigma \models \varphi \wedge \psi && \text{iff } \sigma \models \varphi \wedge \sigma \models \psi \\
&\sigma \models \neg\varphi && \text{iff } \sigma \not\models \varphi \\
&\sigma \models \mathsf{X}\,\varphi && \text{iff } \sigma_{1\ldots} \models \varphi \\
&\sigma \models \varphi\ \mathsf{U}\ \psi && \text{iff } \exists j \geq 0.\ \sigma_{j\ldots} \models \psi \wedge \forall 0 \leq i < j.\ \sigma_{i\ldots} \models \varphi.
\end{aligned}
$$

B Algorithm to Identify Accepting Runs

Algorithm 1 presents the DFS-based procedure from Jackermeier and Abate [22] for computing paths from a given LDBA state to accepting cycles. This yields a set of accepting runs that we then convert to sequences of Boolean formulae. We employ two practical optimisations during this conversion process: first, as observed by Jackermeier and Abate [22], requiring the policy to avoid *all* other LDBA states $q \notin \{q_i, q_{i+1}\}$ at step i can be overly restrictive. Instead, we construct β_i^- to only avoid assignments that lead to: (i) sink states (states from which satisfying the specification is impossible), or (ii) previous states on the current path (which would undo progress). This relaxation allows the policy more flexibility while still preventing catastrophic transitions. Second, when a positive formula β_i^+ contains a disjunction, we generate a separate formula sequence for each disjunct in β_i^+. This avoids requiring the policy to internally select between the different options. Instead, we employ the value function to choose the approximately optimal formula sequences as described in Section 4.4.

C Experimental Details

C.1 Implementation

Our implementation is built in JAX [8]. We use Rabinizer 4 [28] to convert LTL specifications into LDBAs. All experiments were run on an NVIDIA GeForce RTX 4070 GPU; we were able to train 10 seeds of a given algorithm in parallel to completion within roughly 30 minutes for ZoneEnv-NM, and roughly 3 hours in Warehouse.

C.2 Environments

ZoneEnv-NM. The ZoneEnv environment was introduced by Vaezipoor et al. [50]. It consists of a walled plane with 8 circular regions ("zones") that have four different colours and form the atomic propositions. Since the zones never overlap, the assignments all correspond to singleton sets of atomic propositions (or the empty set). It features a 2-dimensional continuous action space for acceleration and steering. The environment features a high-dimensional state space based on lidar information about the zones, and 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, the episode is terminated prematurely. To increase the complexity of the environment, we introduce ZoneEnv-NM, which expands upon ZoneEnv to introduce state-modifying behaviour; when the agent first enters a purple zone, all green zones disappear for the rest of the episode. The agent must learn to reason about this behaviour in order to successfully complete complex tasks. A visualisation of the ZoneEnv-NM environment is provided in Figure 3a.

Warehouse. The warehouse environment consists of a continuous state space, on which a point agent moves via continuous actions. The agent can interact with two types of objects: crates and vases. Both the agent position and the object positions are randomly sampled at the beginning of each episode. There are furthermore three different fixed regions in the grid: region A (red), region B (green), and door (purple). The agent observes objects via a lidar sensor, and additionally receives directional information about the regions. The action space is composite: at each step, the agent can apply both a continuous movement action and pick up or drop a crate or vase. The atomic

Algorithm 1 Computing paths to accepting cycles [22]

Require: LDBA $B = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{E}, \mathcal{F})$ and current state q.

function DFS(q, p, i) {i is index of last seen accepting state, or -1}
  $P \leftarrow \emptyset$
  if $q \in \mathcal{F}$ **then**
    $i \leftarrow |p|$
  end if
  for all $a \in \Sigma \cup \{\varepsilon\}$ **do**
    $p' \leftarrow [p, q]$
    $q' \leftarrow \delta(q, a)$
    if $q' \in p$ **then**
      if index of q' in $p \leq i$ **then**
        $P \leftarrow P \cup \{p'\}$
      end if
    else
      $P \leftarrow P \cup$ DFS(q', p', i)
    end if
  end for
  return P
end function
{Main entry point}
if $q \in \mathcal{F}$ **then**
  $i \leftarrow 0$
else
  $i \leftarrow -1$
end if
return DFS$(q, [\,], i)$

Table 4: Possible assignments in the Warehouse environment.

{region_a}	{region_b}	{door}
{vase}	{crate}	{region_a, vase}
{region_a, crate}	{region_b, vase}	{region_b, crate}
{door, vase}	{door, crate}	{vase, crate}
{region_a, vase, crate}	{region_b, vase, crate}	{door, vase, crate}
	$\emptyset$	

propositions specify whether the agent is currently carrying a vase or a crate, and if the agent is in one of the three regions. Table 4 lists all possible combinations of propositions that hold true at some state in the Warehouse environment. See Figure 3b for a visualisation of the environment.

C.3 Training Curricula

We develop a generic training curriculum for each environment, detailed in Table 5. Each curriculum stage consists of a distribution over sequences of Boolean formulae in DNF. Later curriculum stages generally consist of more difficult tasks. Once the agent has achieved sufficient success on tasks sampled from the current stage (as measured by successes over the last 256 episodes), we move on to the next stage. We define four general, environment-independent sets of formulae used to construct the curricula: PROPS are simply formulae of the form p, where $p \in AP$. AND consists of conjunctions of propositions up to length 3. ANDNOT consists of formulae of the form $p_1 \wedge \neg p_2$. Finally, OR consists of disjunctions of all possible formulae from the previous sets. We furthermore distinguish two sequence types: REACH-AVOID sequences of the form $((\beta_1^+, \beta_1^-), \ldots, (\beta_n^+, \beta_n^-))$ and REACH-STAY sequences $((\beta_\varepsilon, \beta_1^-), (\varphi, \neg\varphi), (\varphi, \neg\varphi), \ldots)$ that contain the ε-token β_ε and require the agent to at some point hold formula φ for n time steps. For DeepLTL we compute satisfying assignments for each formula in order to obtain a sequence of assignments, rather than a sequence of formulae.

Table 5: Curricula for training sequences. For each stage, we list the success rate threshold κ for progression (measured over a rolling window of the 256 latest episodes), the sequence type (reach-avoid/reach-stay), the probability $\mathrm{P}_{\mathrm{seq}}$ of sampling a sequence of that type, the number of steps n in the sequence (or, for reach-stay tasks, the number of consecutive time steps for satisfying the "stay" sub-task), the type of each reach (β_i^+) and avoid (β_i^-) Boolean formula, and the probability $\mathrm{P}_{\mathrm{avoid}}$ of sampling avoid formulae at each timestep. Dashes (—) indicate "not applicable".

	Stage	κ	Type	$\mathrm{P}_{\mathrm{seq}}$	n	β_i^+	β_i^-	$\mathrm{P}_{\mathrm{avoid}}$
ZoneEnv-NM	1	0.9	Reach-Avoid	1.0	1	props	—	0.0
	2	0.95	Reach-Avoid	1.0	2	props	—	0.0
	3	0.95	Reach-Avoid	1.0	1	props	props	1.0
	4	0.9	Reach-Avoid	1.0	2	props	props	1.0
	5	0.9	Reach-Avoid	0.4	[1, 2]	props	props+ORs	0.7
			Reach-Stay	0.6	30	props	props	0.5
	6	0.9	Reach-Avoid	0.8	[1, 2]	props	props+ORs	0.7
			Reach-Stay	0.2	60	props	props	0.5
	7	—	Reach-Avoid	0.8	[1, 2]	props	props+ORs	0.7
			Reach-Stay	0.2	60	props	props+ORs	0.5
Warehouse	1	0.9	Reach-Avoid	1.0	1	props	—	0.0
	2	0.95	Reach-Avoid	1.0	1	all except ORs	—	0.0
	3	0.9	Reach-Avoid	1.0	2	all except ORs	—	0.0
	4	0.95	Reach-Avoid	1.0	1	all except ORs	all except ORs	0.5
	5	0.95	Reach-Avoid	1.0	[1,2]	all except ORs	all except ORs	0.5
	6	0.9	Reach-Avoid	0.4	[1, 2]	all except ORs	all	0.5
			Reach-Stay	0.6	30	all excepts ORs	all except ORs	0.2
	7	—	Reach-Avoid	0.8	[1, 2]	all except ORs	all	0.5
			Reach-Stay	0.2	60	all except ORs	all	0.5

LTL2Action does not originally employ curriculum learning, but we experimented with various training curricula for a fair comparison. We found it important to include diverse LTL formulae even in early curriculum stages (in contrast to the curricula for the other methods), since LTL2Action operates directly on the syntax level and hence requires exposure to all the various LTL operators. In ZoneEnv-NM, we sample formulae of the reach-avoid family of the form $\neg \mathsf{a}\ \mathsf{U}\ ((\mathsf{b} \vee \mathsf{c}) \wedge \ldots)$ with increasing depth and number of disjunctions in later curriculum stages. We use the same family for the Warehouse curriculum, but furthermore sample Boolean formulae from the AND and ANDNOT families instead of simple atomic propositions in later stages. Note that LTL2Action does not support infinite-horizon specifications, and we hence do not sample formulae containing the G operator.

C.4 Hyperparameters

Table 6 lists hyperparameters of StructLTL and the baselines. We report PPO parameters and details on the neural network architectures used for processing environment observations and the actor and critic. ZoneEnv-NM hyperparameters were chosen to match those of ZoneEnv in previous work [50, 22] and adapted for the Warehouse environment. We use the Adam optimiser [24] throughout.

Model architectures. All methods employ the same overall model architecture. Environment observations are encoded with an MLP and the current LTL task is encoded in a method-dependent way (see below). The resulting latent representations are then concatenated and fed through actor and critic MLPs, respectively. The critic directly outputs a scalar estimate of the value function, whereas the actor outputs the parameters of the action distribution. In ZoneEnv-NM, the actor consists of an MLP with two output heads for the mean and standard deviation of a diagonal Gaussian. In Warehouse, the actor MLP additionally produces logits for a categorical distribution over the discrete

Table 6: Hyperparameters for LTL2Action, DeepLTL, StructLTL, and GenZ-LTL across ZoneEnv-NM and Warehouse. Values spanning multiple columns indicate identical settings across those methods/environments. GenZ-LTL uses a Safe PPO variant, which introduces additional hyperparameters that are not applicable (N/A) to the other methods.

Category	Hyperparameter	ZoneEnv-NM				WarehouseEnv			
		LTL2Action	DeepLTL	StructLTL	GenZ-LTL	LTL2Action	DeepLTL	StructLTL	GenZ-LTL
PPO	Total timesteps	— 2×10^7 —				— 2×10^8 —			
	Environments	— 16 —				64	— 128 —		
	Steps/update	— 4096 —				— 1024 —			
	Minibatches	— 32 —				— 4 —			
	Update epochs	— 10 —				— 5 —			
	Discount (γ)	— 0.998 —							
	GAE lambda (λ)	— 0.95 —							
	Clip epsilon	— 0.2 —							
	Entropy coef.	— 0.003 —							
	Value func. coef.	— 1.0 —				— 0.5 —			1.0
	Learning rate	— 3×10^{-4} —				— 5×10^{-4} —			
	Max grad norm	— 0.5 —				— 1.0 —			
	Adam epsilon	— 10^{-8} —							
Safe PPO	Cost discount (γ_c)	— N/A —			0.998	— N/A —			0.998
	Cost value func. coef.	— N/A —			1.0	— N/A —			1.0
	Lagrangian coef.	— N/A —			1.0	— N/A —			1.0
	Target cost	— N/A —			-0.05	— N/A —			-0.1
	Min Lagrangian	— N/A —			10^{-2}	— N/A —			10^{-3}
	Max Lagrangian	— N/A —			5.0	— N/A —			3.0
Curriculum	Episode window	— 256 —							
Env Net	Hidden sizes	— [128] —							
	Output size	— 64 —							
	Activation	— Tanh —							
Actor	Hidden sizes	— [64, 64, 64] —							
	Activation	— ReLU —							
Critic	Hidden sizes	— [64, 64] —							
	Activation	— Tanh —							
Environment	Max episode length	— 1,000 —							

actions. Finally, for StructLTL and DeepLTL the actor MLP additionally has a head that outputs the log-probability of taking an ε-action.

LTL encoding networks. LTL2Action encodes the LTL formula using a relational graph convolutional network (RGCN) [45] applied to the formula's syntax tree. Following the original paper, it consists of 8 layers with tanh activations, using an embedding dimension of 32 for ZoneEnv-NM and 16 for the Warehouse environment. The weights of all layers are shared, and each layer receives as input the output from the previous layer concatenated with the original one-hot encodings of the symbols in the syntax tree.

DeepLTL encodes sequences of sets of assignments using 16-dimensional embeddings (one for each assignment) processed by a DeepSets module [54] with a hidden layer size of 32 and an output size of 16, using ReLU activations and summation as the aggregation mechanism. The encoded reach and avoid sets are then concatenated and processed with a gated recurrent unit (GRU) [13] with embedding size 32. The GRU is applied back-to-front, and the final hidden state forms the sequence embedding. We use $k = 2$ (the number of repeating loops at test time) throughout.

StructLTL uses 16-dimensional proposition embeddings and processes clauses and disjuncts with separate DeepSets networks, both with no hidden layers, output size 16, and ReLU activations. The attention mechanism uses hidden dimension $d_k = 32$ and ALiBi slope $m = 0.5$ in ZoneEnv-NM, and $m = 5$ in Warehouse. As in DeepLTL, we set the number of accepting cycles before truncation to 2.

GenZ-LTL does not directly use an LTL encoding network. Instead, it employs an *observation reduction* mechanism that modifies the given observation s_t based on the current reach-avoid subgoal extracted from the LTL specification. In ZoneEnv-NM, this mechanism produces 37-dimensional features (proprioceptive information and reach/avoid lidars). However, observation reduction cannot be applied in Warehouse, since it relies on two assumptions: (i) observations must decompose into *identical* proposition-specific components, and this decomposition must be manually known and specified a priori, and (ii) a suitable, manually specified fusion operator is required to handle multiple avoid subgoals. Since neither of these assumptions hold in Warehouse, GenZ-LTL falls back

Table 7: Non-myopic reasoning experiment.

Formula	DeepLTL	GenZ-LTL	StructLTL
$\mathsf{F}\,(\mathsf{purple}) \wedge \mathsf{F}\,(\mathsf{green})$	$0.90^{+0.05}_{-0.05}$	$0.51^{+0.02}_{-0.02}$	$\mathbf{0.95}^{+0.02}_{-0.02}$
$\mathsf{F}\,((\mathsf{purple} \vee \mathsf{red}) \wedge \mathsf{F}\,\mathsf{green})$	$0.77^{+0.23}_{-0.23}$	$0.49^{+0.03}_{-0.03}$	$\mathbf{0.95}^{+0.02}_{-0.02}$
$\mathsf{F}\,\mathsf{purple} \wedge \mathsf{F}\,(\mathsf{red} \wedge (\mathsf{F}\,(\mathsf{yellow} \wedge (\neg\mathsf{red}\ \mathsf{U}\ \mathsf{green}))))$	$0.47^{+0.23}_{-0.23}$	$0.13^{+0.02}_{-0.02}$	$\mathbf{0.81}^{+0.08}_{-0.08}$
$\neg\mathsf{red}\ \mathsf{U}\ (\mathsf{yellow} \wedge \mathsf{F}\,\mathsf{green}) \wedge \mathsf{F}\,\mathsf{purple}$	$0.58^{+0.22}_{-0.22}$	$0.25^{+0.03}_{-0.03}$	$\mathbf{0.84}^{+0.05}_{-0.05}$

to encoding reach-avoid subgoals using a one-hot bitvector representation (see [18, Section 4.1]). The modified observations are then processed by the environment module and rest of the network architecture as in the other methods. We denote this variant as GenZ-LTL†.

C.5 Evaluation Tasks

Table 9 lists the complete LTL specifications used in our evaluation. The ZoneEnv-NM tasks are adapted from established specifications for ZoneEnv from the literature [22], and designed with the non-myopic behavior of ZoneEnv-NM in mind. Warehouse tasks were designed to encompass a broad range of interesting behaviour (such as transporting objects to different regions) and LTL structures. Furthermore, Table 10 lists the evaluation specifications used in the compositional generalisation experiment.

D Additional Results

D.1 Non-myopic Reasoning Experiment

We further validate the non-myopic reasoning capabilities of StructLTL. We investigate the formulae listed in Table 7 in ZoneEnv-NM. Note that all of these formulae require the agent to visit both purple and green; the policy hence has to non-myopically reason about the entire specification to determine that it *first* needs to visit green before visiting purple. Subgoal-decomposition based approaches such as GenZ-LTL are incapable of performing such reasoning.

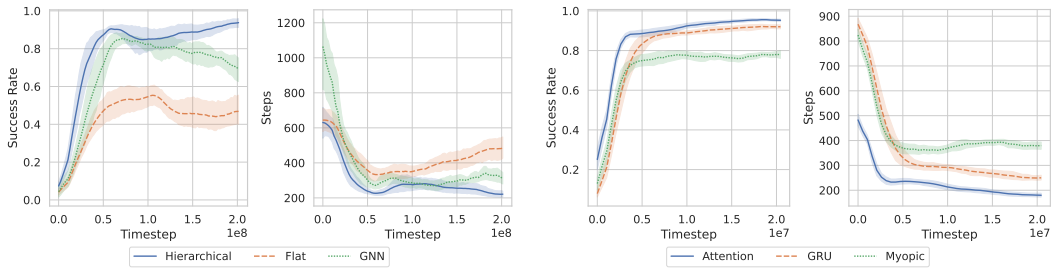
The results in Table 7 clearly show that StructLTL is able to reason non-myopically about the entire task. For example, when attempting to solve $\mathsf{F}\,((\mathsf{purple} \vee \mathsf{red}) \wedge \mathsf{F}\,\mathsf{green})$, the policy realises that it has to achieve green at a later step, and hence tries to avoid purple in the first step (in 95% of cases). As expected, StructLTL performs much better than the myopic baseline GenZ-LTL on these tasks, and we also observe that it outperforms DeepLTL. These results further validate the design of our temporal attention mechanism.

D.2 Ablation Studies

D.2.1 Impact of Hierarchical DNF Encoder

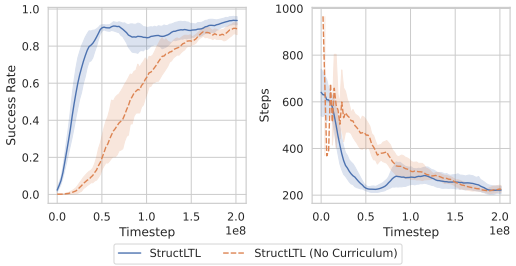
In this ablation study, we investigate the impact of our proposed hierarchical DNF encoder. In particular, we replace the DNF encoder with (i) a flat sequence model that processes Boolean formulae as lists of tokens and (ii) a generic GNN operating on syntax trees. For the sequence model, we use a gated recurrent unit (GRU) [13] with a token embedding size of 16, as in our main experiments. The GNN is instantiated as a graph convolutional network (GCN) [25] with embedding dimension 16, three layers (matching the maximum syntax tree height), and ReLU activations; the final representation of the formula is the final embedding of the root node. Note that we only replace the encoding of Boolean formulae themselves, in order to precisely ablate this part of our proposed architecture; the higher-level sequence encoding is still accomplished with our proposed attention mechanism.

The results in Figure 7a show the performance on Warehouse. They clearly demonstrate that our hierarchical DNF encoding scheme is superior to both flat token-based and GNN encodings of the Boolean formula. While token-based encodings show some success, the policy fails to learn useful representations as it progresses through the curriculum and encounters more complex Boolean formulae. The GNN initially learns quickly, but the representations fail to remain stable throughout training. In contrast, our hierarchical encoding mechanism achieves high success rates.



(a) Impact of the hierarchical DNF encoder

(b) Impact of the temporal attention mechanism



(c) Impact of curriculum learning

Figure 7: Ablation studies. We report averages over finite-horizon evaluation tasks, computed over 50 episodes per task. Shaded areas indicate 95% confidence intervals over 10 random seeds.

D.2.2 Impact of Temporal Attention Mechanism

We proceed to investigate the impact of the temporal attention mechanism. We compare our proposed approach to (i) a myopic baseline that only conditions on the first step of the Boolean formula sequence, and (ii) encoding the sequence of Boolean formulae with a GRU instead of attention. We conduct this ablation study in the ZoneEnv-NM environment.

Figure 7b shows that our proposed attention mechanism performs better than the alternatives. The myopic baseline initially learns quickly, but then plateaus. The GRU converges above the myopic baseline, but below the attention method.

D.3 Impact of Curriculum Learning

Finally, we analyse the impact of curriculum learning. We compare training with our proposed curriculum on WarehouseEnv to training only on formula sequences sampled from the final curriculum stage. Figure 7c shows that the curriculum significantly improves sample efficiency and leads to increased final performance. However, our proposed architecture is still able to learn a successful policy when not guided by a curriculum.

E Complexity Analysis

Let $n = |AP|$, $|\varphi|$ be the formula length, T the sequence length, and M the total number of literals in the formula sequence. Like other automata-based methods (e.g. DeepLTL, GenZ-LTL), we rely on constructing Büchi automata, which is doubly exponential in $|\varphi|$ in the worst case. The complexity of the DFS to find all paths to accepting cycles depends on the number of simple paths from the current state (see [1] for a detailed analysis). Constructing Boolean formulae using the Quine-McCluskey algorithm is exponential in n in the worst case, but in practice restricted by the number of *valid* assignments (e.g. the agent cannot be in both region_a and region_b at the same time). In terms of automaton size, formula construction is linear, since we need to construct one formula per transition in the worst case.

Table 8: Mean wall-clock time (ms) over 100 trials for each processing stage on WarehouseEnv.

Stage	Depth 2	Depth 4	Depth 8	Depth 12
LDBA construction	354.1	373.5	398.7	421
Boolean formula generation	3.0	3.2	6.3	12.8
StructLTL forward pass	0.2	0.2	0.2	0.2

Crucially, all of this is precomputed once per test specification rather than at every environment step. Once the formulae have been constructed, the DNF encoder is linear in M and the attention operation is linear in T (not quadratic, since only the first step performs attention to the other steps).

Table 8 shows the mean wall-clock time for each processing stage in WarehouseEnv across different sequence depths, averaged over 100 trials. The LDBA construction dominates the preprocessing time, and Boolean formula generation scales linearly with the number of transitions, as expected.

F Trajectory Visualisations

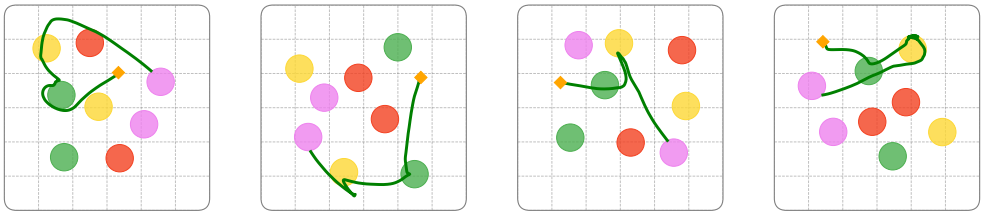
Figures 8 and 9 show trajectories produced by StructLTL on various specifications in ZoneEnv-NM and Warehouse, respectively. The trajectories were obtained by executing the final policy at the end of training conditioned on the given tasks, and confirm that the policy correctly satisfies diverse specifications.

Table 9: Complete list of evaluation specifications.

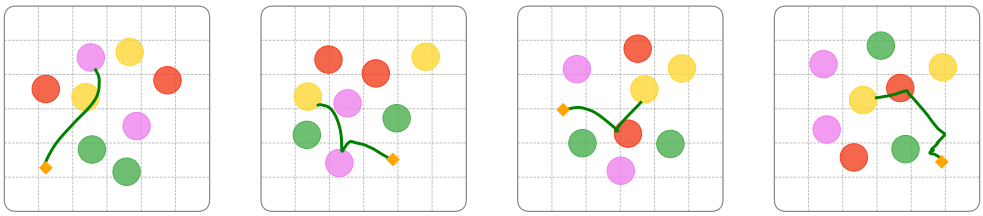
ID	Environment	LTL Formula
		Finite Horizon
φ_1	ZoneEnv-NM	$(\mathsf{F}\,\mathsf{red}) \wedge (\neg\mathsf{red}\;\mathsf{U}\;(\mathsf{green} \wedge \mathsf{F}\,\mathsf{yellow}))$
φ_2	ZoneEnv-NM	$\mathsf{F}\,(\mathsf{red} \vee \mathsf{green}) \wedge \mathsf{F}\,\mathsf{yellow} \wedge \mathsf{F}\,\mathsf{purple}$
φ_3	ZoneEnv-NM	$\neg(\mathsf{purple} \vee \mathsf{yellow})\;\mathsf{U}\;(\mathsf{red} \wedge \mathsf{F}\,\mathsf{green})$
φ_4	ZoneEnv-NM	$\neg\mathsf{green}\;\mathsf{U}\;((\mathsf{red} \vee \mathsf{purple}) \wedge (\neg\mathsf{green}\;\mathsf{U}\;\mathsf{yellow}))$
φ_5	ZoneEnv-NM	$((\mathsf{green} \vee \mathsf{red}) \rightarrow (\neg\mathsf{yellow}\;\mathsf{U}\;\mathsf{purple}))\;\mathsf{U}\;\mathsf{yellow}$
φ_6	ZoneEnv-NM	$\mathsf{F}\,(\mathsf{green} \wedge (\neg\mathsf{red}\;\mathsf{U}\;\mathsf{yellow})) \wedge \mathsf{F}\,\mathsf{purple}$
φ_7	ZoneEnv-NM	$\mathsf{F}\,(\mathsf{yellow} \vee \mathsf{purple} \vee \mathsf{red}) \wedge (\neg\mathsf{yellow}\;\mathsf{U}\;\mathsf{green})$
φ_8	ZoneEnv-NM	$\mathsf{F}\,\mathsf{purple} \wedge \mathsf{F}\,\mathsf{green} \wedge \mathsf{F}\,\mathsf{yellow} \wedge \mathsf{F}\,\mathsf{red}$
φ_9	Warehouse	$\mathsf{F}\,(\mathsf{region_a} \wedge \mathsf{F}\,(\mathsf{crate} \wedge \mathsf{F}\,\mathsf{region_b}))$
φ_{10}	Warehouse	$\neg(\mathsf{region_a} \vee \mathsf{door})\;\mathsf{U}\;(\mathsf{vase} \wedge \mathsf{region_b})$
φ_{11}	Warehouse	$\neg\mathsf{door}\;\mathsf{U}\;(\mathsf{region_a} \wedge (\neg\mathsf{vase}\;\mathsf{U}\;\mathsf{region_b}))$
φ_{12}	Warehouse	$\neg(\mathsf{vase} \vee \mathsf{crate})\;\mathsf{U}\;(\mathsf{region_a} \wedge \mathsf{F}\,(\mathsf{door} \wedge \mathsf{F}\,\mathsf{region_b}))$
φ_{13}	Warehouse	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{region_a} \wedge (\mathsf{region_a}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{region_a})))$
φ_{14}	Warehouse	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \neg\mathsf{crate} \wedge \mathsf{region_b})))$
φ_{15}	Warehouse	$(\mathsf{crate} \rightarrow (\mathsf{crate}\;\mathsf{U}\;\mathsf{region_b}))\;\mathsf{U}\;(\mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{crate} \wedge \mathsf{region_b})))$
φ_{16}	Warehouse	$(\mathsf{vase} \rightarrow (\mathsf{vase}\;\mathsf{U}\;\mathsf{door}))\;\mathsf{U}\;(\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{crate} \wedge \mathsf{region_b}))) \wedge \mathsf{F}\,(\mathsf{vase} \wedge \mathsf{door} \wedge (\mathsf{door}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{door}))))$
φ_{17}	Warehouse	$(\mathsf{crate} \rightarrow (\neg(\mathsf{region_a} \vee \mathsf{region_b}))\;\mathsf{U}\;\mathsf{door})\;\mathsf{U}\;(\mathsf{vase} \wedge \mathsf{door} \wedge (\mathsf{door}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{door})))$
φ_{18}	Warehouse	$\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{region_a} \wedge (\mathsf{region_a}\;\mathsf{U}\;(\neg\mathsf{crate} \wedge \mathsf{region_a}))) \wedge \mathsf{F}\,(\mathsf{vase} \wedge \mathsf{door} \wedge (\mathsf{door}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{door})))$
		Infinite Horizon
ψ_1	ZoneEnv-NM	$\mathsf{G}\,\mathsf{F}\,\mathsf{red} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{green} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{yellow}$
ψ_2	ZoneEnv-NM	$\mathsf{F}\,((\mathsf{red} \vee \mathsf{purple}) \wedge \mathsf{F}\,\mathsf{yellow}) \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{green} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{red}$
ψ_3	ZoneEnv-NM	$\mathsf{F}\,(\mathsf{red} \wedge \mathsf{F}\,(\mathsf{green} \vee \mathsf{purple})) \wedge \mathsf{G}\,(\mathsf{red} \rightarrow \mathsf{F}\,(\mathsf{green} \wedge \mathsf{F}\,\mathsf{yellow})) \wedge \mathsf{G}\,(\mathsf{yellow} \rightarrow \mathsf{F}\,\mathsf{red})$
ψ_4	ZoneEnv-NM	$\mathsf{F}\,\mathsf{G}\,\mathsf{red}$
ψ_5	ZoneEnv-NM	$\mathsf{F}\,\mathsf{G}\,\mathsf{red} \wedge \mathsf{F}\,(\mathsf{yellow} \wedge \mathsf{F}\,\mathsf{green})$
ψ_6	ZoneEnv-NM	$\mathsf{F}\,\mathsf{G}\,\mathsf{purple} \wedge \mathsf{G}\,\neg\mathsf{yellow}$
ψ_7	ZoneEnv-NM	$\mathsf{G}\,((\mathsf{green} \vee \mathsf{yellow}) \rightarrow \mathsf{F}\,\mathsf{red}) \wedge \mathsf{F}\,\mathsf{G}\,(\mathsf{green} \vee \mathsf{purple})$
ψ_8	ZoneEnv-NM	$\mathsf{F}\,(\mathsf{purple} \wedge \mathsf{F}\,(\mathsf{green} \vee \mathsf{red})) \wedge \mathsf{F}\,\mathsf{G}\,\mathsf{yellow}$
ψ_9	Warehouse	$\mathsf{G}\,\mathsf{F}\,\mathsf{region_a} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{region_b} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{door}$
ψ_{10}	Warehouse	$\mathsf{G}\,\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{F}\,\neg\mathsf{vase}) \wedge \mathsf{G}\,\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{F}\,\neg\mathsf{crate}) \wedge \mathsf{G}\,\neg\mathsf{region_a}$
ψ_{11}	Warehouse	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{region_a} \wedge (\mathsf{region_a}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{region_a}))) \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{door} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{region_b}$
ψ_{12}	Warehouse	$\mathsf{G}\,\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{region_a} \wedge (\mathsf{region_a}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{region_a}))) \wedge \mathsf{G}\,\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{crate} \wedge \mathsf{region_b})))$
ψ_{13}	Warehouse	$\mathsf{F}\,\mathsf{G}\,\mathsf{region_a}$
ψ_{14}	Warehouse	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \mathsf{region_b}))) \wedge \mathsf{F}\,\mathsf{G}\,\mathsf{region_b}$
ψ_{15}	Warehouse	$\mathsf{F}\,\mathsf{G}\,(\mathsf{crate} \wedge \mathsf{region_a})$
ψ_{16}	Warehouse	$\mathsf{F}\,\mathsf{G}\,(\mathsf{vase} \wedge \mathsf{region_b}) \wedge \mathsf{G}\,\neg\mathsf{region_a} \wedge \mathsf{G}\,\neg\mathsf{crate}$
ψ_{17}	Warehouse	$\mathsf{G}\,((\mathsf{door} \vee \mathsf{region_b}) \rightarrow \mathsf{F}\,\mathsf{region_a}) \wedge \mathsf{F}\,\mathsf{G}\,(\mathsf{vase} \vee \mathsf{crate})$
ψ_{18}	Warehouse	$\mathsf{F}\,\mathsf{G}\,\mathsf{crate} \wedge \mathsf{G}\,(\mathsf{vase} \rightarrow \mathsf{F}\,(\mathsf{door} \wedge \neg\mathsf{crate}))$

Table 10: Evaluation specifications used in the compositional generalisation experiment.

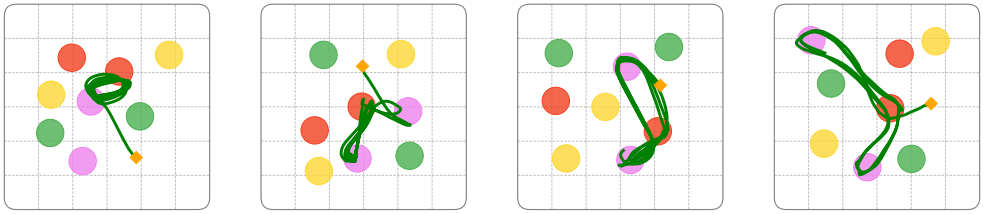
ID	LTL Formula
φ_{19}	$\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{vase} \wedge \mathsf{region_a})$
φ_{20}	$\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{vase} \wedge \mathsf{region_b})$
φ_{21}	$\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{vase} \wedge \mathsf{door})$
φ_{22}	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{crate} \wedge \mathsf{region_a} \wedge (\mathsf{region_a}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \neg\mathsf{crate} \wedge \mathsf{region_a})))$
φ_{23}	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \neg\mathsf{crate} \wedge \mathsf{region_b})))$
φ_{24}	$\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{crate} \wedge \mathsf{door} \wedge (\mathsf{door}\;\mathsf{U}\;(\neg\mathsf{vase} \wedge \neg\mathsf{crate} \wedge \mathsf{door})))$



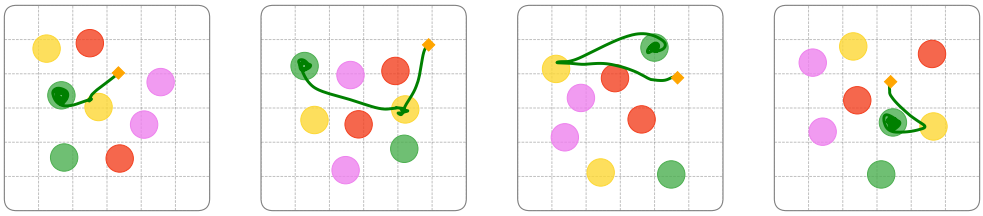
(a) $\mathsf{F}\,(\text{green} \wedge (\neg\text{red}\ \mathsf{U}\ \text{yellow})) \wedge \mathsf{F}\,\text{purple}$



(b) $\mathsf{F}\,(\text{red} \vee \text{purple}) \wedge \mathsf{F}\,\text{yellow} \wedge \mathsf{G}\,\neg\text{green}$

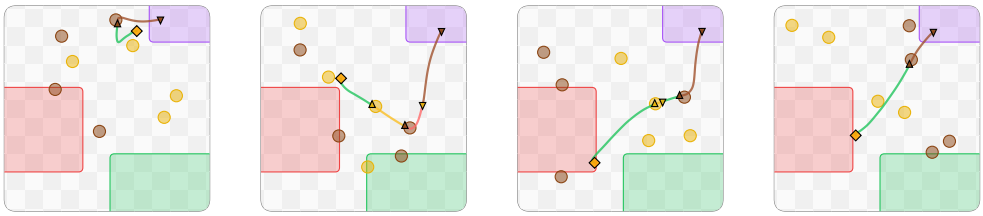


(c) $\mathsf{G}\,\mathsf{F}\,\text{purple} \wedge \mathsf{G}\,\mathsf{F}\,\text{red} \wedge \mathsf{G}\,\neg\text{yellow}$

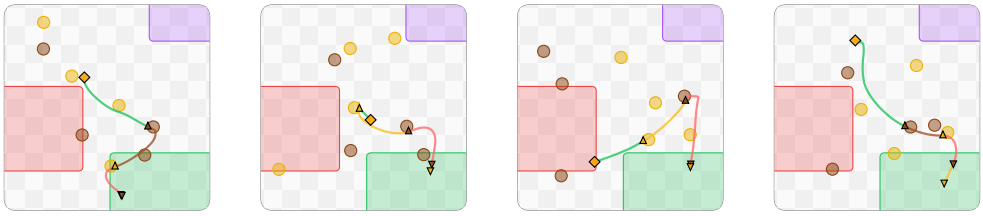


(d) $\mathsf{F}\,\text{yellow} \wedge \mathsf{F}\,\mathsf{G}\,\text{green} \wedge \mathsf{G}\,(\text{purple} \rightarrow \mathsf{F}\,\text{red})$

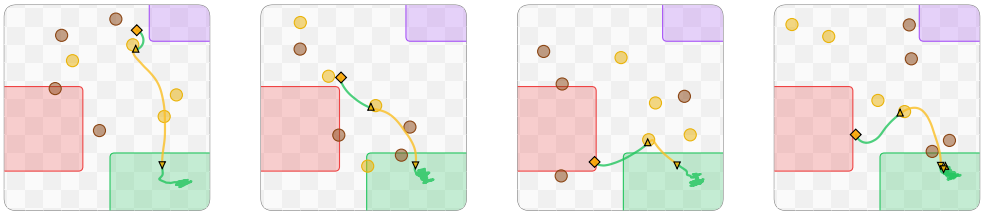
Figure 8: Example trajectories of StructLTL in ZoneEnv-NM. The orange diamond indicates the starting position of the agent. All trajectories were generated by the same policy after training.



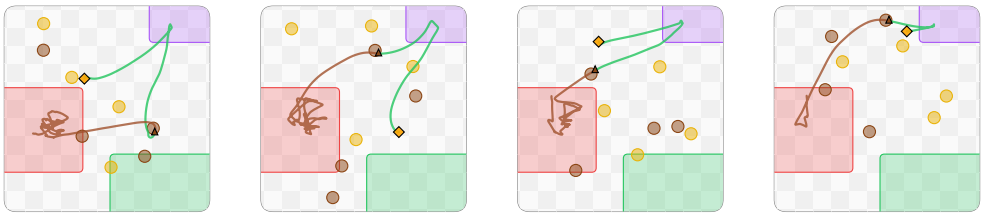
(a) $\mathsf{F}\,(\mathsf{crate} \wedge \mathsf{door} \wedge (\mathsf{door}\ \mathsf{U}\ (\neg\mathsf{crate} \wedge \mathsf{door})))$



(b) $\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{crate} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\ \mathsf{U}\ (\neg\mathsf{vase} \wedge \neg\mathsf{crate} \wedge \mathsf{region_b})))$



(c) $\mathsf{F}\,(\mathsf{vase} \wedge \mathsf{region_b} \wedge (\mathsf{region_b}\ \mathsf{U}\ (\neg\mathsf{vase} \wedge \mathsf{region_b}))) \wedge \mathsf{F}\,\mathsf{G}\,\mathsf{region_b}$



(d) $\mathsf{F}\,\mathsf{door} \wedge \mathsf{F}\,\mathsf{G}\,(\mathsf{crate} \wedge \mathsf{region_a})$

Figure 9: Example trajectories of StructLTL in the Warehouse environment. The orange diamond indicates the starting position of the agent, and yellow and brown circles correspond to vases and crates, respectively. A triangle pointing upwards indicates picking up an object of the corresponding colour, while a triangle pointing downwards indicates dropping an object. Different trajectory colours similarly highlight the objects the agent is currently carrying. All trajectories were generated by the same policy after training.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope, including an explicit list of contributions in the introduction. All claims are supported by the experiments presented in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a discussion of limitations in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper includes comprehensive experimental details in Appendix C, and all code used for experiments can be found in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: All code used for experiments can be found in the supplementary material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper includes comprehensive experimental details in Appendix C. These details can also be found in the code provided in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Details on error bars for figures and tables are provided in Section 5, under "Evaluation Protocol". Further technical details can be found in the code provided in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details on compute resources are provided in Appendix C.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have read and followed the NeurIPS Code of Ethics. This work does not involve human subjects, sensitive data, or applications with foreseeable harmful misuse.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper presents foundational research on Machine Learning. It is not tied to any specific application or deployment, and there is no direct path to negative societal impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper poses no such risks. It does not release pre-trained models, image generators, or scraped datasets.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: This paper uses JAX (Apache v2.0) and Rabinizer 4 (GNU GPL v3.0), both cited in Appendix C.1. Neither is bundled with our code; JAX is installed as a Python dependency and Rabinizer is installed separately by the user.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: All code used for experiments is publicly released at github.com/mathiasj33/structltl. It includes documentation and instructions for reproducing all experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this paper does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.