Zero-Shot Instruction Following in RL via Structured LTL Representations

Mattia Giuri [*1] **Mathias Jackermeier** [*1] **Alessandro Abate** [1]

Abstract

Linear temporal logic (LTL) is a compelling framework for specifying complex, structured tasks for reinforcement learning (RL) agents. Recent work has shown that interpreting LTL instructions as finite automata, which can be seen as high-level programs monitoring task progress, enables learning a single generalist policy capable of executing arbitrary instructions at test time. However, existing approaches fall short in environments where multiple high-level events (i.e., atomic propositions) can be true at the same time and potentially interact in complicated ways. In this work, we propose a novel approach to learning a multi-task policy for following arbitrary LTL instructions that addresses this shortcoming. Our method conditions the policy on sequences of simple Boolean formulae, which directly align with transitions in the automaton, and are encoded via a graph neural network (GNN) to yield structured task representations. Experiments in a complex chess-based environment demonstrate the advantages of our approach.

1. Introduction

In recent years, we have seen remarkable progress in training artificial intelligence (AI) agents to follow arbitrary instructions (Luketina et al., 2019; Liu et al., 2022; Paglieri et al., 2025; Klissarov et al., 2025). One of the central considerations in this line of work is which type of instruction should be provided to the agent; while many works focus on tasks expressed in natural language (Goyal et al., 2019; Hill et al., 2020; Carta et al., 2023), there recently has been increased interest in training agents to follow instructions specified in *formal* language (Jothimurugan et al., 2021; Vaezipoor et al., 2021; Qiu et al., 2023; Yalcinkaya et al., 2024; Jackermeier & Abate, 2025). As a type of *programmatic task representation*, formal specification languages offer several desirable properties, such as well-defined semantics and compositionality. This makes formal instructions especially appealing in safety-critical settings, in which we want to define precise tasks with a well-defined meaning, rather than giving ambiguous natural language commands to the agent (León et al., 2021).

One particular formal language that has proven to be a powerful and expressive tool for specifying tasks in reinforcement learning (RL) settings is *linear temporal logic* (LTL; Pnueli, 1977) (Hasanbeig et al., 2018; Hahn et al., 2019; Kuo et al., 2020; Vaezipoor et al., 2021; León et al., 2022; Liu et al., 2024). 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, non-Markovian behavior in a compositional manner, naturally incorporating aspects like safety constraints and long-term goals. Recent work has exploited the connection between LTL and corresponding automata structures (typically variants of Büchi automata; Büchi, 1966), which provide a *programmatic* way to monitor task progress, to train generalist policies capable of executing arbitrary LTL instructions at test time (Qiu et al., 2023; Jackermeier & Abate, 2025).

However, existing approaches often struggle in scenarios where multiple atomic propositions can hold true simultaneously. This is due to the fact that current methods treat possible assignments of propositions *in isolation*, and do not explicitly model the complex interactions that may occur between different high-level events. In this paper, we develop a novel approach that addresses these limitations. Our approach translates transitions in the automaton into equivalent Boolean formulae, which provide succinct, structured representations of the propositions that are relevant for making progress towards the given task, and explicitly capture the logical conditions for the transition. We show that encoding these formulae via a graph neural network (GNN) yields meaningful representations that can be used to train a policy conditioned on different ways of achieving a given task, enabling zero-shot generalization to novel LTL instructions at test time. Our main contributions are as follows:

[*]Equal contribution [1]Department of Computer Science, University of Oxford. Correspondence to: Mathias Jackermeier <mathias.jackermeier@cs.ox.ac.uk>.

ICML 2025 Workshop on Programmatic Representations for Agent Learning, Vancouver, Canada.

- we develop a novel approach to learning policies for following arbitrary LTL instructions that can effectively handle complex interactions between atomic propositions;
- we propose representing LTL instructions as sequences of Boolean formulae, and show how existing policy learning approaches can be augmented with GNNs to improve representation learning;
- we introduce a novel, chess-like environment where many different propositions can be true simultaneously, which allows us to study the performance of existing approaches in this challenging setting;
- lastly, we conduct an extensive empirical evaluation of our proposed method on this challenging environment, and show that it achieves state-of-the art results and outperforms existing methods.

2. Related Work

While significant research effort has explored the use of LTL to specify tasks in RL (Fu & Topcu, 2014; De Giacomo et al., 2018; Hasanbeig et al., 2018; Camacho et al., 2019; Hahn et al., 2019; Bozkurt et al., 2020; Cai et al., 2021; Shao & Kwiatkowska, 2023; Voloshin et al., 2023; Le et al., 2024; Shah et al., 2025), most approaches are limited to training agents to satisfy a *single* specification that is fixed throughout training and evaluation. In contrast, we aim to learn a general policy that can zero-shot execute arbitrary LTL instructions at test time.

Several works have begun to tackle this challenge of learning generalist policies. Early methods, such as that of Kuo et al. (2020), propose composing recurrent neural networks (RNNs) that mirror the structure of LTL formulae, but this approach requires learning a non-stationary policy, which is generally challenging (Vaezipoor et al., 2021). Other approaches decompose LTL tasks into subtasks, which are then completed sequentially by a goal-conditioned policy (Araki et al., 2021; León et al., 2022; Liu et al., 2024; Xu & Fekri, 2024). However, such methods can exhibit myopic behavior, leading to globally suboptimal solutions because they do not consider the full specification structure during subtask execution. In contrast, our method conditions the policy on the entire sequence of Boolean formulae that need to be satisfied in order to complete the task.

Vaezipoor et al. (2021) introduce LTL2Action, which directly encodes the LTL formula's syntax tree using a GNN and employs LTL progression (Bacchus & Kabanza, 2000) to update the task representation. While this allows for generalization to some extent, the primary drawback of this method is that it requires the policy to *learn* the semantics of temporal operators. Instead, we construct Büchi automata to explicitly capture the temporal structure of tasks, and condition the policy on sequences of simple *Boolean* formulae.

Our work builds upon recent approaches that introduced the idea of exploiting the structure of Büchi automata for training a general LTL-conditioned policy (Qiu et al., 2023; Jackermeier & Abate, 2025). However, our work differs in how the task is represented and processed. Instead of learning a policy conditioned directly on atomic propositions (Qiu et al., 2023), or on sequences of sets of assignments (Jackermeier & Abate, 2025), we propose to translate the transitions in the automaton into equivalent Boolean formulae and learn a policy conditioned on sequences thereof. This provides a more explicit and structured representation of the task transition dynamics, especially when multiple propositions can be simultaneously true and interact in complex ways, leading to better generalization and performance.

3. Background

3.1. Reinforcement Learning

We consider a standard reinforcement learning (RL) setup where an agent interacts with an environment, modeled as a Markov decision process (MDP). 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. The agent's goal is to learn a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ (a mapping from states to probability distributions over actions) that maximizes 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 policy π starting from $s_0 \sim \rho_0$, i.e., $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$. The value function of a policy 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 policy π starting in state s.

3.2. Linear Temporal Logic

Linear temporal logic (LTL; Pnueli, 1977) is a modal logic used to specify properties of infinite sequences of states, serving as a formal programmatic specification language. LTL formulae are defined over a set of atomic propositions (AP), which represent basic properties of the environment (e.g., "object A is at location X"). The syntax of LTL is

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

where $\top$ denotes true, $\mathsf{p} \in AP$ is an atomic proposition, $\neg$ (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. Common derived temporal operators include $\mathsf{F}\,\varphi \equiv \top\ \mathsf{U}\ \varphi$ (eventually or finally) and $\mathsf{G}\,\varphi \equiv \neg\mathsf{F}\,\neg\phi$ (globally or always).

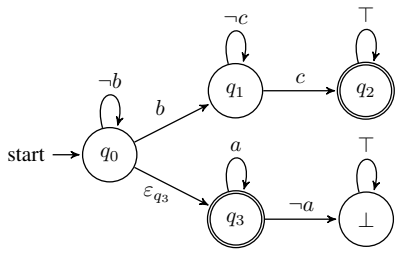

Figure 1. LDBA for the formula $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,(\mathsf{b} \wedge \mathsf{F}\,\mathsf{c})$.

LTL semantics are defined over infinite traces $\sigma = \sigma_0\sigma_1\sigma_2\ldots$, where each $\sigma_t \subseteq \text{AP}$ is an *assignment*, i.e., a set of atomic propositions true at time t. For an MDP, we assume a labeling function $L : \mathcal{S} \to 2^{AP}$ that maps each environment state $s \in \mathcal{S}$ to the set of atomic propositions true in that state. A trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$ satisfies an LTL formula φ, denoted $\tau \models \varphi$, if its corresponding trace $L(s_0)L(s_1)\ldots$ satisfies φ according to LTL semantics. For example, the LTL formula $\neg\mathsf{a}\ \mathsf{U}\ \mathsf{b}$ is satisfied by exactly the traces where eventually b is true at some time t, and a is false at all timesteps before. See Appendix A for a formal definition of LTL satisfaction semantics. LTL provides a formal and structured way to define complex tasks to RL agents, such as "eventually reach region A, and if region B is entered, then eventually reach region C while avoiding region D."

3.3. Büchi Automata for LTL

The semantics of LTL formulae can be captured by Büchi automata (Büchi, 1966), which serve as *explicit, programmatic structures* encoding a particular task. We here focus on limit-deterministic Büchi automata (LDBAs; Sickert et al., 2016), which are defined as tuples $\mathcal{B} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F}, \mathcal{E})$. $\mathcal{Q} = \mathcal{Q}_N \uplus \mathcal{Q}_D$ 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 $\mathcal{F} \subseteq \mathcal{Q}_D$ and $\delta(q, \alpha) \in \mathcal{Q}_D$ for all $q \in \mathcal{Q}_D$ and $\alpha \in \Sigma$. The only way to transition from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ is by taking a jump transition $\varepsilon \in \mathcal{E}$, which does not consume any input. Given an input trace σ, a *run* of $\mathcal{B}$ is an infinite sequence of states in $\mathcal{Q}$ respecting the transition function δ. A trace is *accepted* by $\mathcal{B}$ if there exists a run that infinitely often visits accepting states.

Theorem 3.1 (Sickert et al., 2016)**.** *Given an LTL formula φ, it is possible to automatically construct an LDBA $\mathcal{B}_\varphi$ such that $\mathcal{B}_\varphi$ accepts exactly the traces σ for which $\sigma \models \varphi$.*

Example 3.1. Figure 1 shows an LDBA that accepts traces satisfying $(\mathsf{F}\,\mathsf{G}\,\mathsf{a}) \vee \mathsf{F}\,(\mathsf{b} \wedge \mathsf{F}\,\mathsf{c})$. Accepting runs either reach q_1 from q_0 via b and then q_2 via c, or use the jump transition to reach q_3, after which a must hold true indefinitely.

Büchi automata are appealing structures to represent LTL instructions, since they explicitly capture the memory required to execute a given task. In RL settings, 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 (Hasanbeig et al., 2018; Hahn et al., 2019). 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, updates to the LDBA state affect the behavior of the policy. For example, in Figure 1 the policy initially might aim to make b true (following the upper branch), but once it has transition to state q_1, it would aim to make c true instead.

4. Method

We leverage goal-conditioned RL (Liu et al., 2022) to learn a generalist policy capable of executing arbitrary instructions specified in LTL. Our approach follows a standard framework (Hasanbeig et al., 2018; Vaezipoor et al., 2021; Jackermeier & Abate, 2025): during training, we sample random LTL specifications φ at the beginning of each episode, and generate a trajectory by repeatedly sampling $a_t \sim \pi(s_t \,|\, \varphi)$. We simultaneously keep track of the current state of the LDBA $\mathcal{B}_\varphi$ constructed from φ, assigning positive rewards to actions that lead to accepting states in $\mathcal{B}_\varphi$, and giving a reward of 0 otherwise. Optimizing these rewards amounts to solving the following optimization problem:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi|\varphi}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{\mathcal{F}_{\mathcal{B}_\varphi}}(q_t)]\right],$$

where $\mathbf{1}_A$ is the indicator function of set A. Intuitively, given an LTL instruction φ, the reward-optimal policy will visit accepting states in $\mathcal{B}_\varphi$ as often as possible, and hence satisfy the given specification. For a detailed discussion of how the reward-optimal policy relates to the optimal policy w.r.t. the probability of satisfying a given specification, see (Hahn et al., 2019; Voloshin et al., 2023; Jackermeier & Abate, 2025).

A key challenge in the above framework is how to condition the policy on a given LTL instruction φ. Prior work has shown that directly encoding φ, for example using a recurrent or graph neural network, tends to be ineffective, since this requires learning a complex non-stationary policy due to the non-Markovian nature of LTL tasks (Vaezipoor et al., 2021). Instead, we build on recent approaches that exploit the information contained in $\mathcal{B}_\varphi$ to learn a stationary policy $\pi\colon \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$ conditioned on both the MDP state s and current LDBA state q (Qiu et al., 2023; Jackermeier & Abate, 2025).

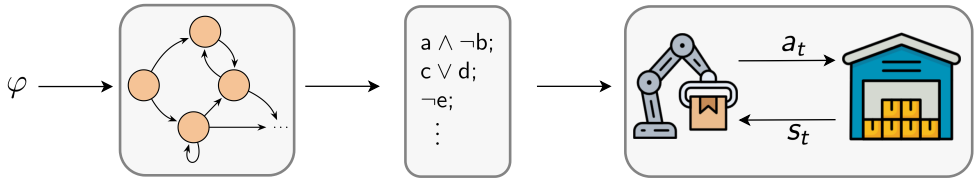

Figure 2. Given an LTL specification φ, we first construct an LDBA $\mathcal{B}_\varphi$ to represent the task structure and memory. We then extract a sequence of Boolean formulae from an accepting run in $\mathcal{B}_\varphi$ to condition our policy. If the LDBA state changes due to progress or external events, we recompute the accepting run from the new state q' and re-condition the policy.

4.1. High-level Overview

Figure 2 illustrates our approach. We first construct an LDBA $\mathcal{B}_\varphi$ from a given LTL instruction φ, which captures the task structure and the memory required to execute it. From the LDBA, we extract a sequence of *Boolean formulae* corresponding to an accepting run from the initial state q_0. Each formula is associated with an edge in $\mathcal{B}_\varphi$, and succinctly represents the conditions that must hold true in order to make progress towards satisfying the task. We then execute a policy conditioned on this sequence of formulae in the environment. If the LDBA state changes to a new state q', either because the policy has made progress towards the task or because of an external event, we recompute an accepting run from q' and condition the policy on the corresponding new sequence of Boolean formulae.

4.2. Extracting Accepting Runs

Given an LDBA $\mathcal{B}_\varphi$ constructed from a specification φ, and a state q, we use Algorithm 1 of (Jackermeier & Abate, 2025) to identify accepting runs starting in q (see Appendix B). This performs a simple depth-first search from q to cycles in the automaton containing at least one accepting state. Each accepting run corresponds to one possible way of achieving the LTL instruction φ.

4.3. Representing Runs via Boolean Formulae

From an accepting run $\rho = (q, q_1, \dots)$ we construct a sequence of Boolean formulae capturing the high-level goals the agent must achieve 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 a_i of atomic propositions that satisfies the transition condition $\delta(q_i, a_i) = q_{i+1}$, while avoiding any transitions to other states (excluding self-loops). We represent this with two Boolean formulae β_i^+ and β_i^- that satisfy

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

where $\mathbb{A} = \{L(s) : s \in \mathcal{S}\} \subseteq 2^{AP}$ is the set of possible assignments in the MDP. Intuitively, β_i^+ is a succint representation of the assignments that allow transitioning from q_i to q_{i+1}, while β_i^- captures the assignments that must not hold true in order to avoid transitioning to other states.

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. 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

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

A succinct representation of the transition from q_0 to q_1 is the formula $\beta_0^+ \equiv a \wedge \neg c$.

Constructing the formulae. To construct one of the formulae β_i^+ and β_i^-, we first identify the set of assignments A for which it must hold true, i.e., the set of assignments corresponding to the relevant LDBA transitions. From this, we can trivially construct a disjunctive normal form (DNF) formula that captures the assignments in A. However, this representation is often not very informative, as it can be large and complex. Instead, we aim to find a small, semantically meaningful formula that captures the essence of the assignments in A while enabling generalization at test time.

In general, this problem of finding a minimal Boolean formula for a given set of assignments is known to be intractable (assuming that $\mathsf{P} \neq \mathsf{NP}$) (Masek, 1979; Allender et al., 2006). We hence employ the following approximate procedure: we initially construct a set of well-structured candidate formulae by combining elementary *formula templates* such as disjunctions of propositions, conjunctions of propositions, conjunctions with negated disjunctions, and combinations thereof. For each formula we construct, we compute its set of satisfying assignments $A \subseteq \mathbb{A}$, and finally for a given set of assignments we choose the associated candidate formula of minimal length. If we encounter a set of assignments for which no precomputed candidate formula exists, we fall back to using the DNF as described above. For further details on constructing candidate formulae, see Appendix D.

4.4. Learning Structured LTL Representations

Having extracted sequences of Boolean formulae from the LDBA, we use a combination of GNNs and RNNs to obtain

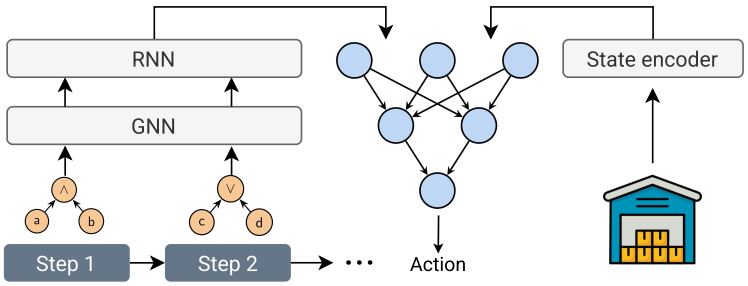

Figure 3. A sequence of Boolean formulae is processed via a GNN to produce a sequence of embeddings, which are passed through an RNN to obtain a representation of an accepting run. The MDP state is encoded via a state encoder. Both embeddings are fed into the policy network, which produces an action.

a meaningful task embedding for the policy. We first transform each formula into an abstract syntax tree (AST), where leaf nodes are propositions and internal nodes are logical operators. We interpret the AST as a directed graph with edges from children to parent nodes. Finally, we associate with each node in the AST a learnable embedding and apply a graph convolutional network (GCN; Kipf & Welling, 2017), which updates the node representations as follows:

$$\mathbf{h}_v^{(l+1)} = f\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_v d_u}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l)}\right),$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v, d_v is the degree of node v in the graph with added self-loops, $\mathbf{h}_v^{(l)}$ is the representation of node v at layer l, $\mathbf{W}^{(l)}$ is a learnable weight matrix, and $f(\cdot)$ is a nonlinearity such as ReLU. The final embedding of the root node is the learned representation of the entire Boolean formula.

To obtain a meaningful representation of an accepting run ρ in the LDBA, we concatenate the embeddings of β_i^+ and β_i^- at each transition and produce an overall embedding by applying a gated recurrent neural network (GRU; Cho et al., 2014). Since ρ is generally an infinite sequence, we consider only a finite prefix of transitions as an approximation.

Policy Architecture. See Figure 3 for an illustration of our overall policy architecture. Given an accepting run in the LDBA, we obtain an embedding from the GNN and RNN as discussed above. We simultaneously encode the current MDP state s using either a multilayer perceptron (MLP) or convolutional neural network (CNN). The policy is instantiated as another MLP that maps from these embeddings to a distribution over actions, i.e., returns the parameters of a categorical or Gaussian distribution.

To handle jump transitions in the LDBA, we follow the standard procedure of augmenting the action space of the policy with designated ε-actions that take the jump transition without performing an action in the MDP (Hasanbeig et al., 2018; Voloshin et al., 2023; Jackermeier & Abate, 2025). In the discrete case, we add an additional output logit to the policy network, and in the continuous case the policy network represents a mixture of a Gaussian distribution for the MDP action space, and a discrete distribution modelling the probability of taking an ε-action.

4.5. Training

We optimize the parameters of our model end-to-end via goal-conditioned RL. Instead of directly sampling LTL formulae during training, we design a training curriculum consisting of increasingly challenging sequences of Boolean formulae to satisfy. Jackermeier & Abate (2025) have previously shown that curriculum learning is an effective method for improving the training of LTL-conditioned policies in practice.

Let $\{(\beta_i^+, \beta_i^-)\}_{i \in [n]}$ be a training sequence sampled from the curriculum. We assign a reward of 1 to an episode if the agent successfully satisfies the formulae β_i^+ in sequence, and assign a negative reward of -1 if the agent instead satisfies the currently active β_i^-. We jointly optimise the policy and learn a value function V^π using proximal policy optimization (PPO; Schulman et al., 2017).

4.6. Selecting an Accepting Run

In order to execute a new LTL instruction φ with our trained model, we extract an accepting run of the Büchi automaton $\mathcal{B}_\varphi$ to condition the policy as described previously. However, in general there are multiple possible accepting runs for any given state of $\mathcal{B}_\varphi$. Similar to previous work (Qiu et al., 2023; Jackermeier & Abate, 2025) we use the *value function* of the policy to select the best accepting run, i.e., the accepting run we are most likely to be able to complete and hence satisfy the task.

Specifically, let s be the current MDP state, q the current LDBA state, $AR = \{\rho_i\}_{i \in [n]}$ be the set of accepting runs computed via Algorithm 1, and ς be the function mapping

a run to a sequence of Boolean formulae. We select the accepting run

$$\rho^* = \arg\max_{\rho \in AR} V^\pi(s, \varsigma(\rho_i))$$

to condition the policy each time the LDBA state is updated.

4.7. Discussion

Representing LTL instructions as sequences of Boolean formulae allows us to learn structured task embeddings that are useful for learning a generalist policy. In particular, Boolean formulae succinctly capture the *meaning* of a (sub-)task and allow the policy to effectively generalize. This is especially true in settings where many atomic propositions can be true at the same time. For example, if the policy has learned how to achieve the formula a and how to achieve the formula b, it can exploit what is has learned about the semantics of Boolean operators to also achieve the formula $\mathsf{a} \wedge \mathsf{b}$. Previous methods (Qiu et al., 2023; Jackermeier & Abate, 2025) do not support this type of generalization, and instead treat different assignments as completely separate goals for the policy.

5. Experiments

We conduct experiments to answer the following questions[1]: **(1)** Can our approach effectively zero-shot generalize to unseen LTL instructions? **(2)** How does our method compare to recent state-of-the-art approaches for training multi-task LTL-conditioned policies? **(3)** How does the performance of our method behave with increasing task difficulty?

5.1. Experimental Setup

ChessWorld. We conduct our experiments in the ChessWorld environment, in which states correspond to positions on an 8×8 chessboard. In the beginning of an episode, the agent — the white king — is randomly placed on an empty square. It then needs to navigate the board by moving along the 8 compass directions in order to reach squares that specific black pieces can move to, while avoiding squares attacked by other pieces. In particular, the atomic propositions correspond to the black pieces, i.e., $AP = \{\mathsf{queen}, \mathsf{rook}, \mathsf{knight}, \mathsf{bishop}, \mathsf{pawn}\}$, and a proposition is true in a square if the corresponding piece is either located on that square or attacks it. Due to the different movement patterns of the pieces, many squares can be attacked by multiple pieces at the same time, leading to a large number of possible assignments and complex interactions between propositions. For an illustration of the environment, see Figure 4. More details about the environment can be found in Appendix C.1.

[1]Our code is available at https://github.com/mattiagiuri/ltl_gnn

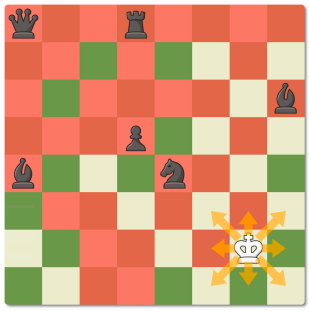

Figure 4. In the ChessWorld environment, the agent (the white king) must navigate the chessboard while avoiding certain squares attacked by black pieces. The agent can move along the 8 compass directions. Blue shading indicates squares attacked by at least one piece.

Tasks. We consider a number of tasks of varying difficulty to evaluate our method. We differentiate between *finite-horizon* tasks, which can be solved in a finite number of steps, and *infinite-horizon* tasks, which specify recurrent behavior that the agent must execute indefinitely. For example, the finite-horizon task $\mathsf{F}\,(\mathsf{queen} \wedge (\neg \mathsf{knight}\ \mathsf{U}\ \mathsf{rook}))$ requires the agent to reach a square attacked by the queen and subsequently avoid squares attacked by the knight until it reaches a square attacked by the rook. In contrast, the infinite-horizon task $\mathsf{F}\,\mathsf{G}\,\mathsf{queen}$ requires the agent to reach squares attacked by the queen and stay there indefinitely. We provide further details on the tasks used in our evaluation in Appendix C.4.

Baselines. We compare our method to DeepLTL (Jackermeier & Abate, 2025), a state-of-the-art approach for learning a generalist policy for following LTL instructions. Similarly to our method, DeepLTL exploits the structure provided by Büchi automata for policy learning. However, it conditions the policy on sets of assignments without explicitly modelling their interactions. In contrast, our method represents the task as a sequence of Boolean formulae, which allows us to learn structured representations with GNNs.

We furthermore compare our approach to a novel baseline that augments DeepLTL with a Transformer encoder (Vaswani et al., 2017) to learn meaningful representations of the sets of assignments. In theory, the attention mechanism of the Transformer should allow this model to learn interactions between the assignments, providing similar advantages as our approach.

Evaluation protocol. All methods are trained with the same training curriculum (see Appendix C.3) for 15M interaction steps. Hyperparameter details are provided in Appendix C.2. We report performance in terms of success rate (SR) and average discounted return $J(\pi)$. All results are averaged over 5 random seeds.

Table 1. Success rates (SR) and average discounted returns $J(\pi)$ on finite-horizon tasks with standard deviations over 5 seeds.

	DeepLTL		Transformer		LTL-GNN	
Task Set	SR	$J(\pi)$	SR	$J(\pi)$	SR	$J(\pi)$
ϕ_1	$99.1_{\pm 1.84}$	$0.906_{\pm 0.017}$	$70.6_{\pm 20.9}$	$0.642_{\pm 0.187}$	$\mathbf{99.3}_{\pm 0.92}$	$\mathbf{0.915}_{\pm 0.009}$
ϕ_2	$92.3_{\pm 3.77}$	$0.886_{\pm 0.037}$	$57.6_{\pm 16.4}$	$0.554_{\pm 0.159}$	$\mathbf{95.2}_{\pm 3.24}$	$\mathbf{0.913}_{\pm 0.029}$
ϕ_3	$68.8_{\pm 3.06}$	$0.658_{\pm 0.030}$	$51.7_{\pm 21.0}$	$0.494_{\pm 0.200}$	$\mathbf{82.6}_{\pm 6.60}$	$\mathbf{0.785}_{\pm 0.061}$
ϕ_4	$91.2_{\pm 2.74}$	$0.886_{\pm 0.027}$	$77.0_{\pm 15.4}$	$0.746_{\pm 0.153}$	$\mathbf{92.7}_{\pm 3.87}$	$\mathbf{0.902}_{\pm 0.036}$
ϕ_5	$67.1_{\pm 5.31}$	$0.642_{\pm 0.050}$	$41.7_{\pm 18.1}$	$0.398_{\pm 0.171}$	$\mathbf{74.3}_{\pm 8.52}$	$\mathbf{0.709}_{\pm 0.078}$
ϕ_6	$91.5_{\pm 2.32}$	$0.873_{\pm 0.023}$	$76.6_{\pm 17.1}$	$0.724_{\pm 0.162}$	$\mathbf{93.6}_{\pm 1.95}$	$\mathbf{0.892}_{\pm 0.016}$
ϕ_7	$\mathbf{91.9}_{\pm 1.34}$	$\mathbf{0.897}_{\pm 0.014}$	$80.8_{\pm 11.1}$	$0.785_{\pm 0.110}$	$91.0_{\pm 3.24}$	$0.888_{\pm 0.031}$

Table 2. Success rates on infinite-horizon tasks with standard deviations over 5 seeds.

Task Set	DeepLTL	Transformer	LTL-GNN
$\phi^{\infty}_{\text{GF}}$	$\mathbf{95.7}_{\pm 01.7}$	$67.6_{\pm 28.5}$	$92.8_{\pm 9.3}$
ϕ^{∞}_1	$40.0_{\pm 49.0}$	$43.1_{\pm 42.2}$	$\mathbf{86.0}_{\pm 28.0}$
ϕ^{∞}_2	$35.7_{\pm 44.0}$	$33.6_{\pm 33.9}$	$\mathbf{76.7}_{\pm 36.7}$

5.2. Results

Tables 1 and 2 show the results of evaluating the trained policies on finite-horizon and infinite-horizon tasks, respectively. We furhermore show the discounted return on finite-horizon tasks over training in Figure 5. We see that our method, denoted as LTL-GNN, is able to effectively generalize to unseen LTL instructions and significantly outperforms the baselines on most tasks, achieving higher success rates and discounted returns. This is particularly evident in the challenging infinite-horizon tasks, where both baselines fail to learn a policy that can consistently satisfy tasks in ϕ^{∞}_1 and ϕ^{∞}_2.

The results demonstrate the advantages of our structured learned task representations based on sequences of Boolean formulae. In particular, the GNN-based representation allows us to learn meaningful task embeddings that capture the interactions between the assignments in the LTL tasks. This is in contrast to the Transformer baseline, which struggles to learn effective representations of the sets of assignments.

Varying the Task Difficulty. We further evaluate the performance of our method and the baselines on tasks of varying difficulty, in which we increase the number of pieces that the agent must avoid in order to complete a task. We consider reach-avoid tasks of the form $\neg \mathsf{a} \,\mathsf{U}\, \mathsf{b}$ and vary the number of pieces that must be avoided from 1 to 5. The results are shown in Figure 6.

As expected, we see that the success rates of all methods decrease with increasing task difficulty, i.e., as the number of pieces to avoid increases. However, our method is able to maintain higher success rates than the baselines even for the most difficult tasks, demonstrating its strong performance on challenging LTL instructions.

6. Conclusion and Future Work

We have presented a novel approach for learning generalist policies for following LTL tasks based on structured task representations. Our method exploits the structure of Büchi automata constructed from a given specification, and extracts sequences of Boolean formulae that succinctly represent different ways of achieving the task. These formulae are encoded by a combination of GNNs and RNNs to condition the policy on the LTL instruction. In contrast to previous methods, this allows us to effectively model interactions between different assignments, and to generalize from simpler to more complicated tasks. We have shown that our approach is able to effectively zero-shot generalize to unseen LTL instructions and outperforms state-of-the-art methods in terms of success rate and discounted return.

There are several interesting directions for future work. It would be interesting to apply our method to larger, more realistic environments with high-dimensional observation and action spaces. In some environments, especially vision-based, the labeling function mapping from observations to atomic propositions may not be known. Future work could explore jointly learning the labeling function or incorporating pre-trained foundation models as high-level event detectors. Generally, these advances could lead to better generalist models capable of following well-defined instructions.

References

Allender, E., Hellerstein, L., McCabe, P., Pitassi, T., and Saks, M. Minimizing DNF formulas and AC^0_d circuits given a truth table. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, 2006.

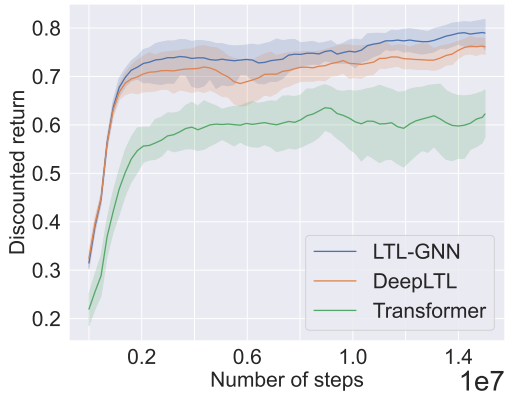

Figure 5. Average discounted returns $J(\pi)$ over the number of interaction steps during training. Shaded areas indicate 90% confidence intervals over 5 random seeds.

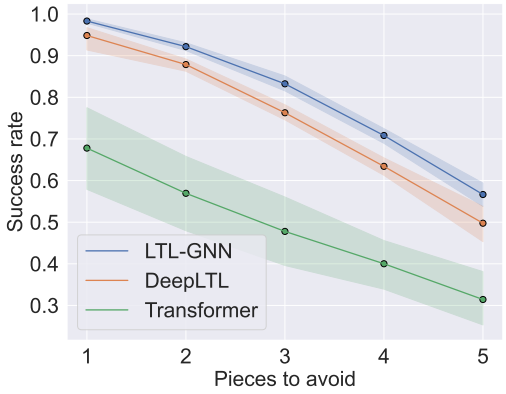

Figure 6. Success rates over tasks with increasing difficulty in terms of the number of pieces to avoid. Shaded areas indicate 90% confidence intervals over 5 random seeds.

Araki, B., Li, X., Vodrahalli, K., Decastro, J., Fry, M., and Rus, D. The Logical Options Framework. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 307–317, 2021.

Bacchus, F. and Kabanza, F. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1):123–191, 2000.

Baier, C. and Katoen, J.-P. *Principles of Model Checking*. MIT Press, 2008.

Bozkurt, A. K., Wang, Y., Zavlanos, M. M., and Pajic, M. Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10349–10355, 2020.

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

Cai, M., Xiao, S., Li, B., Li, Z., and Kan, Z. Reinforcement Learning Based Temporal Logic Control with Maximum Probabilistic Satisfaction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 806–812, 2021.

Camacho, A., Icarte, R. T., Klassen, T. Q., Valenzano, R. A., and McIlraith, S. A. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 6065–6073, 2019.

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y. Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111. Association for Computational Linguistics, 2014.

De Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Reinforcement learning for LTLf/LDLf goals. In *arXiv*, 2018.

Fu, J. and Topcu, U. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*, 2014.

Goyal, P., Niekum, S., and Mooney, R. J. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pp. 2385–2391. AAAI Press, 2019.

Hahn, E. M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., and Wojtczak, D. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 395–412, 2019.

Hasanbeig, M., Abate, A., and Kroening, D. Logically-Constrained Reinforcement Learning. *arXiv*, 2018.

Hill, F., Mokra, S., Wong, N., and Harley, T. Human Instruction-Following with Deep Reinforcement Learning via Transfer-Learning from Text. In *arXiv*, 2020.

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

Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. Compositional Reinforcement Learning from Logical Specifications. In *Advances in Neural Information Processing Systems*, volume 34, pp. 10026–10039, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Klissarov, M., Henaff, M., Raileanu, R., Sodhani, S., Vincent, P., Zhang, A., Bacon, P.-L., Precup, D., Machado, M. C., and D'Oro, P. MaestroMotif: Skill Design from Artificial Intelligence Feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

Kuo, Y.-L., Katz, B., and Barbu, A. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5604–5610, 2020.

Le, X.-B., Wagner, D., Witzman, L., Rabinovich, A., and Ong, L. 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*, 2024.

León, B. G., Shanahan, M., and Belardinelli, F. Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning. In *arXiv*. arXiv, 2021.

León, B. G., Shanahan, M., and Belardinelli, F. In a Nutshell, the Human Asked for This: Latent Goals for Following Temporal Specifications. In *International Conference on Learning Representations*, 2022.

Liu, J. X., Shah, A., Rosen, E., Jia, M., Konidaris, G., and Tellex, S. Skill Transfer for Temporal Task Specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2535–2541, 2024.

Liu, M., Zhu, M., and Zhang, W. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 5502–5511, 2022.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J. N., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 6309–6317, 2019.

Masek, W. J. Some NP-complete set covering problems. *Unpublished manuscript*, 1979.

Paglieri, D., Cupiał, B., Coward, S., Piterbarg, U., Wolczyk, M., Khan, A., Pignatelli, E., Kuciński, Ł., Pinto, L., Fergus, R., Foerster, J. N., Parker-Holder, J., and Rocktäschel, T. BALROG: Benchmarking Agentic LLM and VLM Reasoning On Games. In *The Thirteenth International Conference on Learning Representations*, 2025.

Pnueli, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pp. 46–57, 1977.

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

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv*, (preprint), 2017.

Shah, A., Voloshin, C., Yang, C., Verma, A., Chaudhuri, S., and Seshia, S. A. LTL-Constrained Policy Optimization with Cycle Experience Replay. *Transactions on Machine Learning Research*, 2025.

Shao, D. and Kwiatkowska, M. Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 4180–4189, 2023.

Sickert, S., Esparza, J., Jaax, S., and Křetínský, J. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *Computer Aided Verification*, Lecture Notes in Computer Science, pp. 312–332, 2016.

Vaezipoor, P., Li, A. C., Icarte, R. A. T., and Mcilraith, S. A. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10497–10508. PMLR, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Annual Conference on Neural Information Processing Systems 2017*, pp. 5998–6008, 2017.

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

Xu, D. and Fekri, F. Generalization of temporal logic tasks via future dependent options. *Machine Learning*, 2024.

Yalcinkaya, B., Lauffer, N., Vazquez-Chanlatte, M., and Seshia, S. A. Compositional Automata Embeddings for Goal-Conditioned Reinforcement Learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

A. Semantics of LTL

The satisfaction semantics of LTL are specified by the satisfaction relation $\sigma \models \varphi$, which is recursively defined as follows (Baier & Katoen, 2008):

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

B. Algorithm to Identify Accepting Runs

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

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

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

C. Experimental Details

C.1. ChessWorld

The ChessWorld environment cosists of an 8×8 chessboard. Each state $(x, y) \in [8]^2$ corresponds to a square on the board. The atomic propositions correspond to the black pieces on the chessboard. If piece p is located on square $s = (x, y)$ or p attacks square s, then p is active in s, i.e., $p \in L(s)$. Table 3 lists all combinations of propositions that hold true at some state in ChessWorld. The action space consists of 9 possible actions: the possible 8 moves of a king in chess (i.e., move one square orthogonally or diagonally), and a "stay" action that does nothing.

C.2. Hyperparameters

Neural Networks. In all experiments, our policy is instantiated with a fully connected neural network with dimensions of [128, 64, 64] and ReLU activations. Its output is a categorical distribution, modeled by a softmax layer. The critic network has [128, 64] units with ReLU activations.

Table 3. Possible assignments in ChessWorld.

{queen}	{rook}	{knight}
{bishop}	{pawn}	{queen, rook}
{queen, bishop}	{queen, pawn, bishop}	{queen, pawn, rook}
{knight, rook}	{bishop, rook}	{knight, bishop}

Table 4. PPO hyperparameters.

Parameter	Value
Number of processes	16
Steps per process per update	2048
Epochs	10
Batch size	4096
Discount factor	0.98
GAE-λ	0.95
Entropy coefficient	0.003
Value loss coefficient	0.5
Max gradient norm	0.5
Clipping (ϵ)	0.2
Adam learning rate	0.0003
Adam epsilon	1e-08

We employ a GCN with hidden dimension $d = 32$ and 3 layers. The DeepSets unit in DeepLTL uses a [32, 32] feed-forward network with ReLU activations. The Transformer encoder for the baseline has the following structure: pre-layer normalization, multi-head self-attention with 2 heads and dimension 32, residual connection and layer normalization, residual connection and a [32, 32] feed-forward network.

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

C.3. Training Curriculum

The curriculum consists of three stages, each characterised by a distribution over sequences of Boolean formulae with task difficulty increasing over time. Tasks involve logical combinations (i.e., conjunctions/disjunctions) of atomic propositions, e.g., $\mathsf{bishop} \wedge \mathsf{queen}$, $\mathsf{rook} \vee \mathsf{pawn}$, or $\mathsf{knight} \wedge \neg\mathsf{bishop}$. In the first stage, the focus is on simple reach-only tasks and reach-avoid tasks with at most one piece to avoid. In the second stage, the avoid formulae expand to include up to three pieces, and we introduce more complex finite tasks (e.g., "avoid being attacked until reaching a piece") as well as reach-stay specifications. The third stage further increases the challenge by requiring longer persistence in reach-stay tasks, while keeping the same finite-horizon tasks as the previous stage. For DeepLTL and the Transformer baseline we translate the sequence of Boolean formulae back to a sequence of assignments.

C.4. Experiment Tasks

Tables 5 and 6 list the LTL tasks used in our evaluation.

D. Mapping Sets of Assignments to Boolean Formulae

The construction of the mapping begins with a curated dataset of desirable formulae, denoted as FD. Each formula $\psi \in FD$ is translated into a set of assignments G_ψ, using a mapping that replaces logical operators with set-theoretic counterparts (e.g. $\wedge \rightarrow \cap$). The collection $FA = \{G_\psi : \psi \in FD\}$ constitutes the assignment space corresponding to the formula dataset.

Table 5. ChessWorld finite-horizon LTL specifications.

Task Set	Specifications
ϕ_1	$\mathsf{F}\,(\mathsf{pawn} \wedge \mathsf{F}\,(\mathsf{rook} \wedge \mathsf{F}\,\mathsf{knight}))$ $\mathsf{F}\,((\mathsf{rook} \wedge \mathsf{queen}) \wedge \mathsf{F}\,\mathsf{bishop})$ $\mathsf{F}\,(\mathsf{bishop} \wedge \mathsf{rook}) \wedge \mathsf{F}\,(\mathsf{bishop} \wedge \mathsf{knight})$
ϕ_2	$\neg(\mathsf{pawn} \vee \mathsf{bishop})\ \mathsf{U}\ (\mathsf{bishop} \wedge \mathsf{rook})$ $\neg(\mathsf{queen} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{queen})$ $\neg(\mathsf{bishop} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{knight})$ $\neg(\mathsf{knight} \vee \mathsf{rook})\ \mathsf{U}\ \mathsf{bishop}$ $\neg(\mathsf{bishop} \vee \mathsf{knight})\ \mathsf{U}\ \mathsf{queen}$ $\neg(\mathsf{rook} \vee \mathsf{bishop})\ \mathsf{U}\ \mathsf{pawn}$
ϕ_3	$\neg(\mathsf{bishop} \vee \mathsf{knight} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{queen})$ $\neg(\mathsf{knight} \vee \mathsf{rook} \vee \mathsf{bishop})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{bishop})$ $\neg(\mathsf{bishop} \vee \mathsf{pawn} \vee \mathsf{rook})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{queen})$ $\neg(\mathsf{bishop} \vee \mathsf{knight} \vee \mathsf{queen})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{queen})$
ϕ_4	$\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{knight} \vee \mathsf{pawn})\ \mathsf{U}\ \mathsf{queen}$ $\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{knight} \vee \mathsf{queen})\ \mathsf{U}\ \mathsf{pawn}$ $\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{pawn} \vee \mathsf{queen})\ \mathsf{U}\ \mathsf{knight}$ $\neg(\mathsf{bishop} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen})\ \mathsf{U}\ \mathsf{rook}$ $\neg(\mathsf{rook} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen})\ \mathsf{U}\ \mathsf{bishop}$
ϕ_5	$\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen})\ \mathsf{U}\ (\mathsf{queen} \wedge \mathsf{pawn})$ $\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{knight} \vee \mathsf{queen} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{pawn} \wedge \mathsf{rook})$ $\neg(\mathsf{bishop} \vee \mathsf{rook} \vee \mathsf{pawn} \vee \mathsf{queen} \vee \mathsf{knight})\ \mathsf{U}\ (\mathsf{knight} \wedge \mathsf{bishop})$ $\neg(\mathsf{bishop} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen} \vee \mathsf{rook})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{knight})$ $\neg(\mathsf{rook} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen} \vee \mathsf{bishop})\ \mathsf{U}\ (\mathsf{bishop} \wedge \mathsf{queen})$ $\neg(\mathsf{rook} \vee \mathsf{knight} \vee \mathsf{pawn} \vee \mathsf{queen} \vee \mathsf{bishop})\ \mathsf{U}\ (\mathsf{rook} \wedge \mathsf{queen})$
ϕ_6	$\mathsf{F}\,(\mathsf{queen} \wedge (\neg\mathsf{knight}\ \mathsf{U}\ \mathsf{rook}))$ $\neg(\mathsf{pawn} \vee \mathsf{knight})\ \mathsf{U}\ (\mathsf{queen} \wedge \mathsf{rook}) \wedge \mathsf{F}\,\mathsf{pawn}$ $\neg(\mathsf{bishop} \vee \mathsf{rook})\ \mathsf{U}\ \mathsf{pawn} \wedge \mathsf{F}\,\mathsf{knight}$ $\mathsf{F}\,(\mathsf{rook} \wedge (\neg\mathsf{bishop}\ \mathsf{U}\ \mathsf{pawn}))$ $(\neg\mathsf{queen}\ \mathsf{U}\ \mathsf{pawn}) \wedge (\neg\mathsf{bishop}\ \mathsf{U}\ \mathsf{knight})$ $(\neg\mathsf{queen}\ \mathsf{U}\ \mathsf{rook}) \wedge (\neg\mathsf{knight}\ \mathsf{U}\ \mathsf{queen})$ $(\neg\mathsf{queen}\ \mathsf{U}\ \mathsf{pawn}) \wedge (\neg\mathsf{bishop}\ \mathsf{U}\ \mathsf{knight}) \wedge (\neg\mathsf{knight}\ \mathsf{U}\ \mathsf{rook})$
ϕ_7	$\neg(\mathsf{rook} \vee \mathsf{bishop} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{knight} \wedge \neg\mathsf{rook})$ $\neg\mathsf{queen}\ \mathsf{U}\ (\mathsf{bishop} \wedge \neg\mathsf{pawn})$ $\neg(\mathsf{bishop} \vee \mathsf{knight})\ \mathsf{U}\ (\mathsf{queen} \wedge \neg\mathsf{knight})$ $\neg(\mathsf{rook} \vee \mathsf{knight} \vee \mathsf{queen} \vee \mathsf{pawn})\ \mathsf{U}\ (\mathsf{bishop} \wedge \neg\mathsf{queen})$ $\neg(\mathsf{pawn} \vee \mathsf{queen} \vee \mathsf{rook} \vee \mathsf{knight} \vee \mathsf{bishop})\ \mathsf{U}\ (\mathsf{rook} \wedge \neg\mathsf{bishop})$

An injective mapping $FC : FA \to FD$ is then established, ensuring each assignment set is uniquely linked to an optimal formula under a chosen metric. The mapping is designed to include typical LTL-derived assignments; unseen sets during deployment are processed by generating their DNF.

D.1. Notation and Conventions

Let the set of environment variables be denoted by V. We define $P(V)$ as the power set of V, excluding the empty set and V itself. For fixed cardinality, $P(V)_k$ refers to the subset of $P(V)$ containing only sets of size k, i.e., $\binom{V}{k}$. More generally, we define $P(V)_x^y$ as the collection of sets of size y drawn from $P(V)_x$, i.e., $\binom{P(V)_x}{y}$. Each element in $P(V)_x^y$ is thus a set of y subsets of V, each of cardinality x.

Table 6. ChessWorld infinite-horizon LTL specifications.

Task Set	Specifications
ϕ^{∞}_{GF}	$\mathsf{G}\,\mathsf{F}\,\mathsf{knight} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{queen}$ $\mathsf{G}\,\mathsf{F}\,\mathsf{pawn} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{rook}$ $\mathsf{G}\,\mathsf{F}\,\mathsf{bishop} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{knight} \wedge \mathsf{G}\,\neg\mathsf{rook}$ $\mathsf{G}\,\mathsf{F}\,\mathsf{rook} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{pawn} \wedge \mathsf{G}\,\neg\mathsf{knight}$
ϕ^{∞}_{1}	$\mathsf{F}\,\mathsf{G}\,\mathsf{bishop}$ $\mathsf{F}\,\mathsf{G}\,\mathsf{queen}$ $\mathsf{F}\,\mathsf{G}\,\mathsf{rook}$ $\mathsf{F}\,\mathsf{G}\,\mathsf{pawn}$ $\mathsf{F}\,\mathsf{G}\,\mathsf{knight}$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{queen} \vee \mathsf{bishop})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{rook} \vee \mathsf{queen})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{knight} \vee \mathsf{pawn})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{bishop} \vee \mathsf{knight})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{rook} \vee \mathsf{pawn})$
ϕ^{∞}_{2}	$\mathsf{F}\,\mathsf{G}\,(\mathsf{bishop} \wedge \neg\mathsf{rook})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{knight} \wedge \neg\mathsf{bishop})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{queen} \wedge \mathsf{pawn})$ $\mathsf{F}\,\mathsf{G}\,(\mathsf{rook} \wedge \mathsf{queen})$

The mapping is built by composing unions, intersections, and differences of sets $G_a = \{\alpha \,|\, \alpha \in \mathbb{A}, a \in \alpha\}$, where $a \in AP$. It is implemented as a dictionary that maps sets of assignments to logical formulae. The core strategy is to define and insert a curated sequence of *templates*—canonical forms of formulae—into the mapping. Since multiple formulae ψ_i may represent the same set of assignments G, the insertion order of templates reflects increasing syntactic complexity, ensuring that G is represented by the simplest possible formula.

Importantly, for every formula added to the mapping, a corresponding complement formula (representing $\mathbb{A} \setminus G$) is also inserted. Throughout, range(a, b) denotes the set of integers in $[a, b-1]$. Each of the following subsections defines a specific *template* in the order in which they are added to the mapping.

D.2. No Assignments

The empty set of assignments is added at the start, in order to prevent edge cases (i.e., formulae representing empty assignment sets are not added).

D.3. *Or* Formulae

These formulae are of the form

$$\bigvee_{v \in \mathcal{V}} v \quad \forall\, \mathcal{V} \in P(V)_k, \quad k = 1, 2, .., |V| - 1$$

with corresponding complement formulae

$$\neg \left(\bigvee_{v \in \mathcal{V}} v \right) \quad \forall\, \mathcal{V} \in P(V)_k, \quad k = 1, 2, .., |V| - 1$$

These are added to the mapping in increasing order of k, i.e. for $k = 1, 2, .., |V| - 1$.

D.4. *And* Formulae

These formulae are of the form

$$\bigwedge_{v\in\mathcal{V}} v \quad \forall\, \mathcal{V} \in P(V)_k, \quad k = 1, 2, .., |V| - 1$$

with corresponding complement formulae

$$\neg\left(\bigwedge_{v\in\mathcal{V}} v\right) \quad \forall\, \mathcal{V} \in P(V)_k, \quad k = 1, 2, .., |V| - 1$$

These are added to the mapping in increasing order of k, i.e. for $k = 1, 2, .., |V| - 1$.

D.5. *Or-x-and-y* Formulae

Here x and y are parameters to be chosen, in our experiments $x = 4, y = 2$. The formulae are

$$\left(\bigvee_{v\in\mathcal{V}_1} v\right) \wedge \left(\bigwedge_{w\in\mathcal{V}_2} w\right) \quad \begin{cases} \forall\, \mathcal{V}_1 \in P(V)_i, & \text{for } i \in \text{range}(2, x+1) \\ \forall\, \mathcal{V}_2 \in P(V)_j \text{ st } \mathcal{V}_2 \cap \mathcal{V}_1 = \emptyset, & \text{for } j \in \text{range}(1, y+1), \end{cases}$$

added to the mapping in lexicographical order of (i, j). The complement formulae are

$$\neg\left[\left(\bigvee_{v\in V_i} v\right) \wedge \left(\bigwedge_{w\in V_j} w\right)\right].$$

D.6. *And-x-and-not-y* Formulae

Here x and y are parameters to be chosen, in our experiments $x = 2, y = 3$. The formulae are

$$\left(\bigwedge_{v\in\mathcal{V}_1} v\right) \wedge \neg\left(\bigvee_{w\in\mathcal{V}_2} w\right) \quad \begin{cases} \forall\, \mathcal{V}_1 \in P(V)_i, & \text{for } i \in \text{range}(2, x+1) \\ \forall\, \mathcal{V}_2 \in P(V)_j \text{ st } \mathcal{V}_2 \cap \mathcal{V}_1 = \emptyset, & \text{for } j \in \text{range}(1, y+1), \end{cases}$$

added to the mapping in lexicographical order of (i, j). The complement formulae are

$$\neg\left(\bigwedge_{v\in\mathcal{V}_1} v\right) \vee \left(\bigvee_{w\in\mathcal{V}_2} w\right)$$

D.7. *Or-x-and-not-y* Formulae

Here x and y are parameters to be chosen, in our experiments $x = 4, y = 4$. The formulae are

$$\left(\bigvee_{v\in\mathcal{V}_1} v\right) \wedge \neg\left(\bigvee_{w\in\mathcal{V}_2} w\right) \quad \begin{cases} \forall\, \mathcal{V}_1 \in P(V)_i, & \text{for } i \in \text{range}(1, x+1) \\ \forall\, \mathcal{V}_2 \in P(V)_j \text{ st } \mathcal{V}_2 \cap \mathcal{V}_1 = \emptyset, & \text{for } j \in \text{range}(1, y+1), \end{cases}$$

added to the mapping in lexicographical order of (i, j). The complement formulae have the form

$$\neg\left(\bigvee_{v\in\mathcal{V}_1} v\right) \vee \left(\bigvee_{w\in\mathcal{V}_2} w\right)$$

D.8. *Or-x-and-not-zy* Formulae

Here x, y andz are parameters to be chosen, in our experiments $x = 4, y = 3, z = 2$.

$$\left(\bigvee_{v\in\mathcal{V}_1} v\right) \wedge \neg\left[\bigvee_{\mathcal{V}_2\in P(V)_j^k} \left(\bigwedge_{w\in\mathcal{V}_2} w\right)\right] \quad \begin{cases} \forall\, \mathcal{V}_1 \in P(V)_i, & \text{for } i \in \text{range}(1, x+1) \\ \forall\, \mathcal{V}_2 \in P(V)_j^k, & \text{for } j \in \text{range}(2, y+1), k \in \text{range}(1, z+1), \end{cases}$$

added to the mapping in lexicographical order of (i, j, k). The complement formulae are

$$\neg \left(\bigvee_{v \in \mathcal{V}_1} v \right) \vee \left[\bigvee_{\mathcal{V}_2 \in P(V)^k_j} \left(\bigwedge_{w \in \mathcal{V}_2} w \right) \right].$$

D.9. Choosing a Formula

In general there may be sets of assignments with multiple matching formulae, that is, the mapping FC need not be injective. In this case, we select the first matching formula added to the mapping, which implicitly corresponds to selecting the minimal formula in terms of number of operators, i.e., we select

$$\psi^*_G = \arg \min_{\psi \in F_G} c(\psi),$$

where $c(\psi)$ measures the complexity of formula ψ.