Generalization of Compositional Tasks with Logical Specification via Implicit Planning

Duo Xu (✉) and Faramarz Fekri

Georgia Institute of Technology,
Atlanta GA 30332, USA
dxu301@gatech.edu

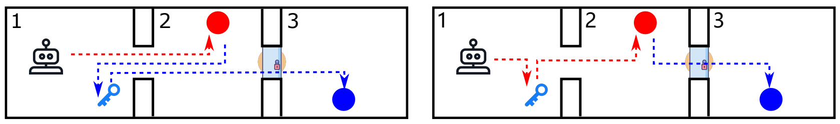
Abstract. In this study, we address the challenge of learning generalizable policies for compositional tasks defined by logical specifications. These tasks consist of multiple temporally extended sub-tasks. Due to the sub-task inter-dependencies and sparse reward issue in long-horizon tasks, existing reinforcement learning (RL) approaches, such as task-conditioned and goal-conditioned policies, continue to struggle with slow convergence and sub-optimal performance in the generalization of compositional tasks. To overcome these limitations, by decomposing the given task into reach-avoid sub-tasks, we introduce a new hierarchical RL framework that trains a high-level planner to select optimal sub-tasks and zero-shot generalizes to other tasks in the sub-task level, which enhances the efficiency and optimality of task generalization. At the high level, we present an implicit planner specifically designed for generalizing compositional tasks. This planner selects the next sub-task and estimates the multi-step return for completing the remaining task from the current state. It learns a latent transition model and performs planning in the latent space to select sub-tasks based on a graph neural network (GNN). Subsequently, the sub-task assigned by the high level guides the low-level module to effectively handle long-horizon tasks, while the estimated return encourages the low-level policy to account for future sub-task dependencies, enhancing its optimality and densifying the sparse rewards. We conduct comprehensive experiments to demonstrate the framework's advantages over previous methods in terms of both efficiency and optimality.

1 Introduction

In real-world applications, such as robotics and control system, task completion often involves achieving multiple subgoals that are spread over time and must follow user-specified temporal order constraints. For instance, a service robot on a factory floor may need to gather components in specific sequences based on the product being assembled, all while avoiding unsafe conditions. These complex tasks are defined through logic-based compositional languages, which have long been essential for objective specification in sequential decision-making [7].

Using domain-specific properties as propositional variables, formal languages like Linear Temporal Logic (LTL) [28] and SPECTRL [13] encode intricate temporal patterns by combining these variables with temporal operators and logical connectives. These languages provide clear semantics and support semantics-preserving transformations into deterministic finite-state automata (DFA), which reveal the discrete structure of an objective to a decision-making agent. Generalizing across multiple tasks is crucial for deploying autonomous agents in various real-world scenarios [33]. In this work, we tackle the problem of generalizing compositional tasks where, at test time, the trained agent is given a DFA description of an unseen task and is expected to accomplish the task without further training.

While reinforcement learning (RL) algorithms have achieved remarkable success across numerous fields [26, 27, 29, 38], they still face challenges in generalizing to compositional tasks, which differ significantly from typical problems addressed by conventional RL methods. Previous works on compositional task generalization [17, 2, 35, 8, 24] trained generalizable agents with satisfying success rate. Some approaches [2, 8, 24] tackle unseen compositional tasks by leveraging trained reusable skills or options. However, these methods train each option to achieve a specific subgoal independently, thereby risking the loss of global optimality in task completion when sub-tasks are dependent on each other. Additionally, methods that train policies directly conditioned on task formulas [17, 35] proposed an effective framework for temporal logic task generalization, which can reach the global optimality in generalization. However, these methods tend to exhibit slow convergence in complex tasks or environments, as they lack task decomposition and miss out on the compositional structure of such tasks. Another recent paper [40] pre-trains embeddings for reach-avoid DFAs of the task, which enables the zero-shot generalization of goal-conditioned RL agent to other tasks. However, this work still treated the given compositional task as a whole and does not address the spare reward issue specifically. Before introducing the proposed framework, we will first present two motivating examples to show the importance of considering sub-task dependencies.



(a) Myopic solution (b) Optimal solution

Fig. 1: Motivating example 1. Task: first go to red ball, and then blue ball. Red: reaching red ball. Blue: reaching blue ball.

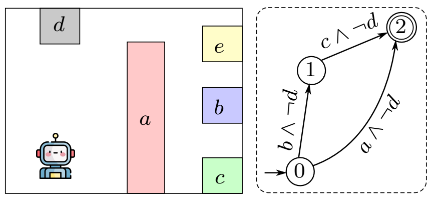


Fig. 2: Motivating example 2. Left: map. Right: task automaton with self loops omitted.

Motivating Examples. The first example, shown in Figure 1, involves a robot starting in room 1 with a locked door between rooms 2 and 3. This door can only be opened using the key located in room 1. The task is to visit the red ball first, then the blue ball. Previous option-based methods train options for reaching each ball independently, disregarding dependencies between subgoals. This approach results in the myopic solution illustrated in Figure 1(a), where the robot wastes additional steps retrieving the key compared to the optimal solution shown in Figure 1(b).

The second example, illustrated in Figure 2 with the task automaton on the right, assigns rewards of 1 and 10 for reaching areas "a" and "c", respectively, while other areas yield no rewards. The optimal solution for completing the task with maximal rewards is to reach "b" first, then "c", while avoiding "a" and "d". However, if task-conditioned policies are trained using previous methods [17, 35, 40], the agent often ends up reaching "a" and avoiding "d" since it is easier and still completes the task, producing sub-optimal solution for the given task. This occurs because the task-conditioned policy is not explicitly trained to avoid "a" while reaching "b" at the initial automaton state 0. The resulting sub-optimality arises from overlooking dependencies of sub-tasks in different branches (i.e. paths to task completion): reaching "a" from the state 0 can directly accomplish the task, making "b" and "c" inaccessible and hence getting sub-optimal rewards for completing the task. In addition, the rewards of achieving some sub-tasks may also be dependent on each other. For example, if the agent first completes the sub-task "b" and then achieves "a", it will get a higher reward than directly achieving "a". This is because the user may prefer some patterns to others in the task specifications.

In this work, in order to address the sub-task dependencies and sparse reward issue in the generalization of compositional tasks, we introduce a hierarchical RL framework which trains a high-level planner to select the optimal reach-avoid sub-task for the low-level module to complete, achieving zero-shot generalization to other tasks in the sub-task level. Unlike previous methods which treat any given compositional task as a whole, our approach decomposes the given task into reach-avoid sub-tasks to resolve the sparse reward issue, and trains a novel implicit planner to select optimal target sub-task for the low-level module to complete by taking sub-task dependencies into consideration. In addition to selecting target sub-tasks, the high-level module also estimates the multi-step return of completing the remaining task for the low-level module, which can guide the low-level module while accounting for dependencies among future sub-tasks. The low-level module is responsible for choosing primitive actions to accomplish the assigned sub-task, with its decisions conditioned on both the current and upcoming sub-tasks required to complete the remaining task.

Specifically, in the high-level module, the implicit planner learns a latent transition model for sub-task transitions, which encodes environmental observations into latent states and predicts the latent state when the input sub-task is complete. By decomposing any given task into sub-tasks and applying a graph neural network (GNN) over predicted latent states of completing sub-tasks, the high-level module generates an embedding vector representing the future situations of completing the remaining task. This embedding vector serves as input to the implicit planner, guiding it to predict the next sub-task and estimate the return for the low-level module. The low-level module is a variant of the agent module in [35], which is conditioned on any sub-task assigned by the high-level module and future sub-tasks to complete.

In experiments, we demonstrate the advantages of the proposed framework over baselines in three environments, including both discrete and continuous state and action spaces. Based on comprehensive experiments, we show the proposed framework outperforms baselines in terms of both optimality and learning efficiency.

2 Preliminary

2.1 Task Specification Language

A compositional task considered in this work is described by a logic specification formula ϕ, a Boolean function that determines whether the objective formula is satisfied by the given trajectory or not [28]. In this work, we adopt the specification language SPECTRL [13] to express the logic and temporal relationships of subgoals in tasks. A specification ϕ in SPECTRL is a logic formula applied to trajectories, determining whether a given trajectory $\zeta = (s_0, s_1, \ldots)$ successfully accomplishes the task specified by ϕ. For rigor of math, ϕ can be described as a function $\phi : \mathcal{Z} \to \{0, 1\}$ producing binary outputs, where $\mathcal{Z}$ is the set of all the trajectories.

Specifically, a specification is defined based on a set of atomic propositions $\mathcal{P}_0$. For each proposition $p \in \mathcal{P}_0$, the MDP state s of the agent satisfies p (denoted as $s \models p$) when $p \in L(s)$ and L is labeling function. The set of symbols $\mathcal{P}$ is composed by conjunctions of atomic propositions in $\mathcal{P}_0$.

Based on definitions above, the grammar for formulating SPECTRL specifications can be written as:

$$\phi ::= \text{ achieve } b \;\mid\; \phi_1 \text{ ensuring } b \;\mid\; \phi_1; \phi_2 \;\mid\; \phi_1 \text{ or } \phi_2 \tag{1}$$

where $b \in \mathcal{P}$. Here "achieve" and "ensuring" correspond to "eventually" and "always" operators in LTL [28, 2]. Given any finite trajectory ζ with length h, the satisfaction of a SPECTRL specification are defined as:

1. $\zeta \models$ achieve b if $\exists i \leq h, s_i \models b$ (or $b \in L(s_i)$)
2. $\zeta \models \phi$ ensuring b if $\zeta \models \phi$ and $\forall i \leq h, s_i \models b$
3. $\zeta \models \phi_1; \phi_2$ if $\exists i < h, \zeta_{0:i} \models \phi_1$ and $\zeta_{i+1;h} \models \phi_2$
4. $\zeta \models \phi_1$ or ϕ_2 if $\zeta \models \phi_1$ or $\zeta \models \phi_2$

Specifically, the statement 1) signifies that the trajectory should eventually reach a state where the symbol b holds true. The statement 2) means that the trajectory should satisfy specification ϕ while always remaining in states where b is true. The statement 3) signifies that the trajectory should sequentially satisfy ϕ_1 and then ϕ_2. The statement 4) says that the trajectory should satisfy either ϕ_1 or ϕ_2. We say a trajectory ζ satisfies specification ϕ if there is a time step h such that the prefix $\zeta_{0:h}$ satisfies ϕ.

In addition, every SPECTRL specification ϕ is guaranteed to have an equivalent directed acyclic graph (DAG), termed as abstract graph [13]. An abstract graph $\mathcal{G}$ is defined as $\mathcal{G} ::= (Q, E, q_0, F, \kappa)$, where Q is the set of nodes, $E \subseteq Q \times Q$ is the set of directed edges, $q_0 \in Q$ denotes the initial node, $F \subseteq Q$ denotes the accepting nodes, subgoal region mapping $\beta : Q \to 2^{\mathcal{S}}$ which denotes the subgoal region for every node in Q, and safe trajectories $\mathcal{Z}_{\text{safe}} = \cap_{e \in E} \mathcal{Z}^{e}_{\text{safe}}$ where $\mathcal{Z}^{e}_{\text{safe}}$ denotes the safe trajectories for any edge $e \in E$. Note that the environmental MDP $\mathcal{M}$ is connected with task specification ϕ and $\mathcal{G}_\phi$ by β and $\mathcal{Z}^{e}_{\text{safe}}$ which may change for different tasks. Furthermore, the function κ labels each edge $e := q \to q'$ with the symbol b_e (labeled edge denoted as $e := q \xrightarrow{b_e} q'$). Given κ, the agent transits from node q to q' when the states s_i and s_{i+l} of trajectory ζ satisfy $s_i \in \beta(q)$ and $b_e \subseteq L(s_{i+l})$ for some $l \geq 0$.

Given a task specification ϕ, the corresponding abstract graph $\mathcal{G}_\phi$ can be constructed based on its definition, such that, for any trajectory $\zeta \in \mathcal{Z}$, we have $\zeta \models \phi$ if and only if $\zeta \models G_\phi$. Hence, the RL problem for task ϕ can be equivalent to the reachability problem for $\mathcal{G}_\phi$. It is obvious that every task DAG has a single initial node in SPECTRL language, which can be converted into a tree.

Sub-task Definition Given the DAG $\mathcal{G}_\phi$ corresponding to task specification ϕ, we can define sub-tasks based on edges of the DAG. Formally, an edge from node q to $p \in Q$ can define a reach-avoid sub-task specified by the following SPECTRL formula:

$$\text{Sub-Task}(q,p) := \text{achieve}(b_{(q,p)}) \text{ ensuring } \left(\bigwedge_{r \in \mathcal{N}(q), r \neq p} \neg b_{(q,r)} \right) \tag{2}$$

where $b_{(q,p)}$ is the propositional formula labeled over the edge (q,p) in the DAG, and $\mathcal{N}(q)$ is the set of neighboring nodes to which the out-going edges of q point in the DAG. For instance, in Figure 2, the propositional formula over the edge (q_0, q_2) is $b_{(q_0,q_2)} = \neg d \wedge a$. When $e = (q,p)$, the notation Sub-Task(e) is same as Sub-Task(q,p) defined in (2), e.g., Sub-Task$(q_0, q_2) :=$ achieve(a)ensuring$(\neg b \wedge \neg d)$ in Figure 2 after some algebra.

For each Sub-Task(q,p) and any MDP state $s_0 \in \mathcal{S}$, there is a policy $\pi_{(q,p)}$ which can guide the agent to produce a trajectory $s_0 s_1 \ldots s_n$ in MDP. It induces the path $qqq \ldots qp$ in the DAG, meaning that the agent's DAG state remains at q until it transits to p, i.e., $s_n \in \beta(p)$ and $s_i \notin \beta(p)$ for $i < n$. In this work, since we consider the dependencies of sub-tasks, the policy $\pi_{(q,p)}$ is also dependent on the future sub-tasks to complete.

Given the environmental MDP M, for any SPECTRL task specification ϕ, the agent first transforms ϕ to its corresponding DAG (abstract graph) $\mathcal{G}_\phi = (Q, E, q_0, F, \beta, \mathcal{Z}_{\text{safe}}, \kappa)$. Then, the sub-tasks of all the edges can be obtained from $\mathcal{G}_\phi$ based on (2). In this work, we assume that for every edge sub-task of the DAG, the achieve part only has conjunction of propositions (denoted as p_+ for reaching) and the ensuring part only has conjunctions of negated propositions (denoted as p_- for avoidance). The propositions in p^+ of sub-task η are regarded as subgoals of η. This is because achieving any negated propositions in the safety condition is assumed infeasible. For example, for the sub-task achieve(b)ensuring$(\neg a \wedge \neg d)$ (i.e., $b \wedge \neg a \wedge \neg d$), we have $p_+ = \{b\}$ and $p_- = \{a, d\}$. Whenever the achieve part in (2) contains disjunction, this sub-task will decomposed further into sub-tasks in parallel edges, until every sub-task only achieves conjunction of propositions.

Remark. For example, based on our assumptions, if the DFA in Figure 2 has $b_{q_0,q_1} = b \wedge \neg d$ and $b_{q_0,q_2} = a \wedge \neg f$, then Sub-Task$(q_0, q_1)$:=achieve(b)ensuring$\neg a \wedge \neg d \wedge \neg f$. Although (2) could yield f in the ensuring part, our assumption ignores the achievement of any propositions which are negated in the safety condition, hence discarding f.

2.2 Problem Formulation

We introduce the labeled MDP as the working environment in Appendix , where propositions and labeling function are defined. Given the labeled MDP M with unknown state transition dynamics and reward function, a SPECTRL specification ϕ represents the logic compositional task consisting of temporally extended sub-tasks, and $\mathcal{G}_\phi$ is the DAG (abstract graph) corresponding to the task ϕ.

The target of this work is to train an reinforcement learning (RL) agent in a data-efficient manner which can be generalized to complete any unseen SPECTRL task ϕ without further training. In addition to task completion, we also consider the optimality of the found solution for the unseen task ϕ, maximizing the discounted accumulated environmental rewards, i.e. return, during task completion. Specifically, the reward function of MDP $\mathcal{M}$ is unknown to the agent, and the reward of any state s is available to the agent only whenever s is visited.

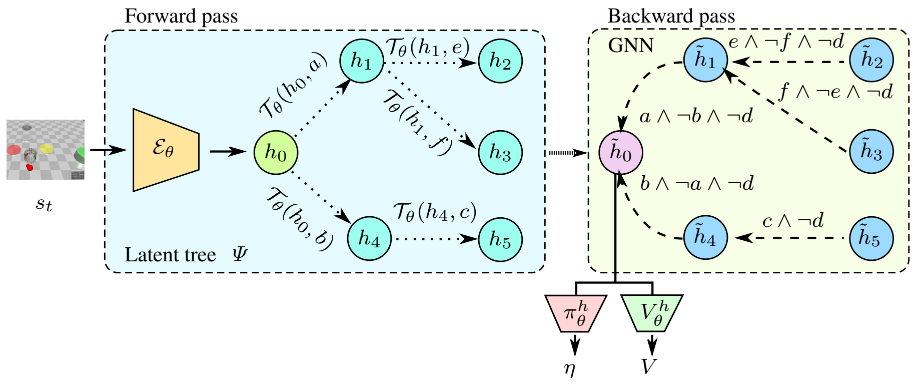

Forward pass
Backward pass
GNN
s_t
$\mathcal{E}_\theta$
h_0
h_1
h_2
h_3
h_4
h_5
$\mathcal{T}_\theta(h_0, a)$
$\mathcal{T}_\theta(h_1, e)$
$\mathcal{T}_\theta(h_1, f)$
$\mathcal{T}_\theta(h_0, b)$
$\mathcal{T}_\theta(h_4, c)$
Latent tree Ψ
$\tilde{h}_0$
$\tilde{h}_1$
$\tilde{h}_2$
$\tilde{h}_3$
$\tilde{h}_4$
$\tilde{h}_5$
$e \wedge \neg f \wedge \neg d$
$f \wedge \neg e \wedge \neg d$
$a \wedge \neg b \wedge \neg d$
$b \wedge \neg a \wedge \neg d$
$c \wedge \neg d$
π_θ^h
V_θ^h
η
V

Fig. 3: Diagram of implicit planner as the high-level agent. The DAG (abstract graph) of the task is shown in the Figure 4. The latent tree is spanned by the encoder $\mathcal{E}_\theta$ and latent transition model $\mathcal{T}_\theta$ in the forward pass, while the feature of future sub-tasks ($\tilde{h}_0$) is extracted by GNN ($\mathcal{M}_\theta$,$\mathcal{U}_\theta$) in the backward pass. The sub-task η and estimated return V are predicted by policy π_θ^h and value networks V_θ^h, respectively, which are realized by MLPs with feature $\tilde{h}_0$ as input. Note that in GNN, every edge is labeled by a corresponding sub-task derived from the task DAG, and the feature of a edge is the binary encoding of positive (p^+) and negative (p_-) propositions of the corresponding sub-task.

3 Methodology

In the following sections, we first detail the modules and operational mechanisms of the proposed framework. Next, we outline the training algorithm, describing the training processes for both low-level and high-level modules. We also propose specific training techniques aimed at enhancing robustness and data efficiency throughout the learning process, including curriculum, experience relabeling and proposition avoidance, which are introduced in Appendix.

3.1 Architecture

The proposed framework consists of high-level and low-level modules. The high-level module is essentially an implicit planner which selects the next sub-task for the low-level agent to complete. Based on the feature of future sub-tasks, the implicit planner is directly trained to predict the best selection of next sub-task and also estimate the return for completing the rest of task, which are passed to the low-level agent for guidance. The low-level module is trained to achieve the assigned sub-task together with the estimated return which makes the low-level policy look into the future sub-tasks. This approach fastens the training of the low-level module, improving the learning efficiency in long-horizon tasks.

High-level Module When the dependencies among sub-tasks are accounted for, the planning problem of selecting the next high-level sub-task no longer adheres to the Markovian property. This limitation prevents the use of the commonly applied value iteration (VI) method for sub-task selection, as VI relies on Bellman equations [32] to compute the value function and is effective only when the Markovian property holds. To address this, we introduce an implicit planner that directly predicts the optimal next sub-task and estimates the expected return for completing the remaining task based on an embedding that represents future sub-tasks and observations. This embedding is generated by a graph neural network (GNN) [30, 41] and a latent transition model [16, 36], described below. As shown in Figure 3, the proposed implicit planner consists of an encoder $\mathcal{E}_\theta$, a latent state transition model $\mathcal{T}_\theta$, a GNN $(\mathcal{M}_\theta, \mathcal{U}_\theta)$, a policy network π^h_θ and a value network V^h_θ. All components of the implicit planner are trained together end-to-end, so their trainable parameters are collectively represented as θ. The implicit planner operates through both forward and backward passes. The task DAG corresponding to Figure 3 is shown in Figure 4.

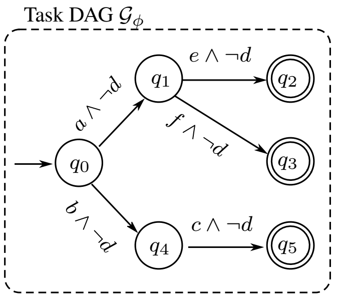


Fig. 4: Task DAG of Figure 3.

Forward Pass. In the forward pass, the planner generates latent representations of the current and future states by using the encoder $\mathcal{E}_\theta$ and the latent dynamic model $\mathcal{T}_\theta$, iteratively constructing a tree Ψ whose node features are predicted latent representations of states. Given the current environmental state s_t, the encoder first derives its latent representation, denoted as $h_0 := \mathcal{E}_\theta(s_t)$, which serves as the root of the latent tree Ψ. Following the structure of the task directed acyclic graph (DAG) $\mathcal{G}_\phi$, the tree Ψ is expanded from h_0 until every accepting node in F of the DAG $\mathcal{G}_\phi$ is included in Ψ.

Expanding Ψ from a node n entails adding all nodes in $\mathcal{G}_\phi$ connected through edges that originate from n. Specifically, the latent state of node n (i.e., h_n) and the subgoals associated with its outgoing edges in $\mathcal{G}_\phi$ are input to the latent transition dynamics model $\mathcal{T}_\theta$, which then predicts the subsequent latent states. These predicted states serve as features for new nodes, which are added to Ψ as the children of node n. This expansion process iterates until all subgoals in $\mathcal{G}_\phi$ are incorporated into Ψ. An example is illustrated in Figure 3. It is worth noting that Ψ is built based on the subgoals (positive propositions of sub-tasks) associated with edges in $\mathcal{G}_\phi$, rather than directly from sub-tasks.

Backward Pass. In the backward pass, the GNN component operates over a graph modified from the latent tree Ψ to extract an embedding vector, $\tilde{h}_0$, which represents the completion of the remaining task, encoding future sub-tasks and predicted states.

Specifically, the direction of each edge in the latent tree Ψ is reversed and labeled with the sub-task which is derived from related edges in the task DAG $\mathcal{G}_\phi$ based on its definition in (2), resulting in a new graph. A multi-layer GNN is then applied to this new graph to extract an embedding vector that encapsulates future sub-tasks and predicted states in the remaining task, denoted by the node feature $\tilde{h}_0$. An example of this process is illustrated in Figure 3. Finally, based on $\tilde{h}_0$, the policy and value function determine the selection of the next sub-task and estimate the return for the low-level agent.

It's important to note that, in the backward pass, the sub-task on each edge of the graph is derived based on (2) and differs from the sub-task in the task DAG $\mathcal{G}_\phi$. This distinction is because the derived sub-task explicitly incorporates conditions to avoid accidentally completing any neighboring sub-tasks, which is not explicitly addressed in the original task DAG $\mathcal{G}_\phi$, as shown in the example of Figure 2.

Encoder. The encoder function, $\mathcal{E}_\theta : \mathcal{S} \to \mathbb{R}^d$, takes an environmental state as input and outputs its latent representation. The encoder's neural architecture is tailored to the environment: a CNN is employed for pixel-based environments, while an MLP is used for environments with continuous observations.

Latent Transition Model. The latent transition function, $\mathcal{T}_\theta : \mathbb{R}^d \times \mathcal{P} \to \mathbb{R}^d$, predicts the next latent state by using the latent of the current state and the subgoals (positive propositions of the sub-task). Given the current state s and sub-task η, the next latent state is predicted as $\mathcal{E}_\theta(s) + \mathcal{T}_\theta(\mathcal{E}_\theta(s), p_\eta^+)$, where p_η^+ is the binary encoding of the subgoals of sub-task η. This function models the changes in the latent state caused by completing a sub-task. In implementation, $\mathcal{T}_\theta$ is typically realized using an MLP.

Graph Neural Network. The GNN is employed to generate an embedding that represents the progress toward completing the remaining tasks. For each node k in the GNN, it first gathers a set of incoming messages from each connected node j with an edge (j, k) directed from j to k. This is achieved using the message-passing function $\mathcal{M}_\theta$, which takes as input the features of nodes k and j ($\tilde{h}_k$ and $\tilde{h}_j$) along with the edge feature $e(j, k)$. The initial feature of each node is its latent state h_k, predicted by $\mathcal{T}_\theta$ during the forward pass, and it is subsequently updated with the incoming messages using the update function $\mathcal{U}_\theta$.

The edge feature $e_{(j,k)}$ is a binary encoding of the sub-task $b_{(k,j)}$ associated with the edge (j, k). Specifically, this feature is created by concatenating two binary vectors that separately represent the positive and negative propositions of the sub-task.

In every layer of GNN, the incoming messages of node k are first aggregated by summation as below,

$$m_k = \bigoplus_{j \in \mathcal{N}(\tilde{h})} \mathcal{M}_\theta(\tilde{h}_j, \tilde{h}_k, e_{(j,k)}) \tag{3}$$

Then, the node feature $\tilde{h}_k$ is updated with incoming message, i.e., $\tilde{h}_k \leftarrow \mathcal{U}_\theta(\tilde{h}_k, m_k)$. In implementation, both $\mathcal{M}_\theta$ and $\mathcal{U}_\theta$ are realized by MLPs.

Since the direction of each edge in Ψ is reversed in the graph used for the backward pass, multiple iterations of applying the functions $\mathcal{M}_\theta$ and $\mathcal{U}_\theta$ in the GNN enable information from each future sub-task to back-propagate to the root node. Consequently, the root node feature, denoted as $\tilde{h}_0$, encapsulates the status of completing future sub-tasks within the remaining task.

Policy and value function. The policy function $\pi^h_\theta : \mathbb{R}^d \to [0,1]^{|\mathcal{P}|}$ maps the embedding $\tilde{h}_0$ extracted by GNN to a distribution of *feasible* next sub-tasks. The next sub-task assigned to the low-level agent is sampled from this output distribution. The value function $V^h_\theta : \mathbb{R}^d \to \mathbb{R}$ maps the embedding $\tilde{h}_0$ to the estimated return for completing the remaining task starting from the current state.

Remark. Previous approaches to planning for logic-based compositional tasks have utilized value iteration [3, 2], Dijkstra's algorithm [10], and heuristic-based search algorithms [15, 9]. These methods rely on the assumption that the cumulative rewards for completing each sub-task are independent of others, meaning the Markovian property must hold. However, as illustrated in the examples in Section 1, this work considers dependencies between sub-tasks, where the cumulative rewards for completing one sub-task depend on future sub-tasks, breaking the Markovian property and making standard planning algorithms inapplicable. To address this, we leverage the generalization capabilities of GNNs to train a value function for sub-tasks and states through supervised learning, enabling the high-level policy to be trained using the PPO algorithm [31].

Low-level Module The target of low-level module is to complete the sub-task η specified by the high-level module, considering the dependencies of future sub-tasks. As discussed in the Sub-task Definition of Section 2.1, every sub-task in SPECTRL language is a reach-avoid task, and can be decomposed into positive proposition (to achieve) and negative propositions (to avoid), stored in the sets p_+ and p_-, respectively. Hence, the low-level policy and value functions, denoted as π^l_ω and V^l_ω, are conditioned on p_+ and p_- encoded into binary vectors. The diagram of processing inputs to the low-level agent is in Figure 5.

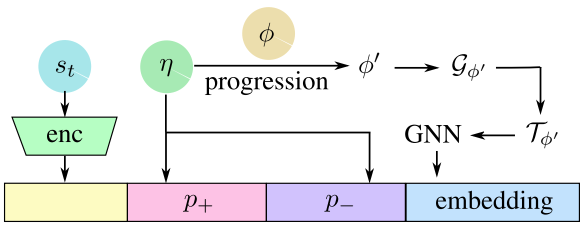


Fig. 5: Diagram of processing inputs to the low-level policy π^l_ω and value function V^l_ω. η is the sub-task assigned by the high-level module. ϕ is the target task to complete. ϕ' is the progression of ϕ with η. The embedding is the representation of ϕ' produced by the GNN. s_t is the environmental observation. "enc" is the encoder mapping the raw observation into a latent vector.

Remark. This low-level module is essentially a variant, where the agent model in [35] is made conditioned on the target sub-task η. Since the high-level module gives an estimate return V upon the completion of every sub-task, the training of the low-level module in this work should be much more efficient than that in [35]. The original model in [35] only has outcome-based reward signals about the whole task.

To account for dependencies of future sub-tasks, both π^l_ω and V^l_ω are conditioned on the DAG of the remaining task ϕ', denoted as $\mathcal{G}_{\phi'}$. Here, ϕ' represents the remainder of the task ϕ after completing sub-task η. Essentially, ϕ' is the progression [19, 35] of task

ϕ once sub-task η is achieved. For example, in the task shown in Figure 2, if sub-task $b \wedge \neg a \wedge \neg d$ (on the edge from state 0 to 1) is completed, the progression of the task becomes $c \wedge \neg d$, indicating the remaining part of the task to be accomplished.

DAG Processing. However, the functions π^l_ω and V^l_ω cannot directly process a DAG. To address this, similar as LTL2Action [35], we use a GNN to generate an embedding from the tree representation of the DAG $\mathcal{G}_{\phi'}$, which can be directly used as input to π^l_ω and V^l_ω. Since any DAG with a single source node can be equivalently converted into a tree, we first convert the DAG of ϕ', denoted $\mathcal{G}_{\phi'}$, into its corresponding tree $\mathcal{T}_{\phi'}$.

This tree $\mathcal{T}_{\phi'}$ differs from the high-level latent tree Ψ. Given that the low-level policy operates at every time step, it's essential to keep the low-level module as streamlined as possible. Thus, the latent transition model is omitted, and each node's initial feature is set to an all-zero vector. Additionally, we ignore the negative propositions of future sub-tasks, using only the binary encoding of the positive propositions (i.e. subgoals) as the edge feature. However, the negative propositions of the current sub-task η are still encoded as a binary vector p_-, as shown in Figure 5.

With these defining node and edge features, a multi-layer GNN is applied to $\mathcal{T}_{\phi'}$, with the direction of each edge reversed. The embedding at the root node then represents the characteristics of all future subgoals to accomplish. The process of obtaining p_+, p_-, and the embedding for future subgoals is illustrated in Figure 5. Note that this low-level GNN differs from the one used at the high level and is trained alongside other low-level components only.

3.2 Algorithm

The high-level and low-level modules are trained independently. In the high level, all components are trained end-to-end to predict the optimal selection of the next sub-task to complete and to estimate the return for completing the remainder of the task. In the low level, the module is trained to accomplish the assigned sub-task while accounting for dependencies from future sub-tasks. To enhance the sample efficiency of the learning process, several training techniques are introduced, including a training curriculum, experience replay, and proposition avoidance. The training of low-level module is introduced in Appendix.

High-level Transition Data. We define a high-level transition tuple as $(s, \phi, \eta, R, s', \phi')$, which indicates that starting from state s, the agent completes the sub-task η and reaches state s'. Here, ϕ' is the progression of task ϕ after sub-task η is completed, representing the remaining part of the task, and R is the accumulated discounted reward earned while completing η. The high-level transition buffer Γ^h stores these transition tuples collected from all trajectories, but only includes transitions where completing the sub-task η resulted in progression over ϕ (i.e., $\phi' \neq \phi$). For a trajectory $\zeta = (s_l, a_l, r_l)_{l=0}^{H-1}$, where K_ζ sub-tasks are completed sequentially, we denote these sub-tasks as $\eta_0, \eta_1, \ldots, \eta_{K_\zeta - 1}$, with the time steps at which they are completed as $t_0, t_1, \ldots, t_{K_\zeta - 1}$. The accumulated rewards obtained for completing each sub-task η_i are defined as $R_i := \sum_{\tau=t_{i-1}}^{t_i} \gamma^{\tau - t_{i-1}} r_\tau$, where r_τ represents the environmental reward at time step τ.

High-level Training The effectiveness of the high-level module depends on the embedding vector that represents the remaining task, which is extracted by the latent transition model and GNN. Therefore, it is essential to train the encoder, latent transition model, and GNN effectively. Drawing inspiration from previous work on learning latent dynamic spaces [4, 16], we utilize the TransE [16] loss to train the encoder $\mathcal{E}_\theta$ and the latent transition function $\mathcal{T}_\theta$ together. For any high-level transition data $(s, \phi, \eta, R, s', \phi')$ and a negatively sampled state $\tilde{s}$, the TransE loss can be expressed as below:

$$\begin{aligned}\mathcal{L}_{\text{TransE}}((s, \eta, s'), \tilde{s}; \theta) = d(&\mathcal{E}_\theta(s) + \mathcal{T}_\theta(\mathcal{E}_\theta(s), \eta), \mathcal{E}_\theta(s')) \\ &+ \max(0, \xi - d(\mathcal{E}_\theta(s), \mathcal{E}_\theta(\tilde{s})))\end{aligned} \tag{4}$$

where θ are the trainable parameters, d is the distance function which is chosen as the Euclidean distance in this work, and ξ is a positive hyper-parameter. The tasks ϕ, ϕ' and reward R are not used in the training loss of $\mathcal{E}_\theta$ and $\mathcal{T}_\theta$.

For the GNN part, models $\mathcal{M}_\theta$ and $\mathcal{U}_\theta$ are trained together with policy π^h_θ and value networks V^h_θ in an end-to-end manner. Since the training curriculum is designed to start from simple tasks, there is no need to pre-train the GNN part.

The components of the high-level module are jointly trained using the PPO algorithm [31] with a set of feasible sub-tasks serving as the action space. Based on the high-level transition buffer Γ^h, the PPO loss is calculated by evaluating the outputs of the policy and value networks through the forward and backward passes, as detailed in Section 3.1. One iteration of training the high-level module can be summarized as the following steps:

1. Sample trajectories ζ from the replay buffer $\mathcal{B}$ which forms the high-level transition dataset Γ^h;
2. Based on transition tuples in Γ^h, compute the PPO [31] and TransE (4) losses, where the negative samples $\tilde{s}$ in (4) are randomly sampled from states in Γ^h;
3. Update parameters θ of all the components in the high-level module together, with gradients of the following loss function:

$$\mathcal{L}(\Gamma^h; \theta) = \mathcal{L}_{\text{PPO}}(\Gamma^h; \theta) + \lambda \sum_i \mathcal{L}_{\text{TransE}}((s_{t_i}, \eta_i, s_{t_{i+1}}), \tilde{s}_i; \theta) \tag{5}$$

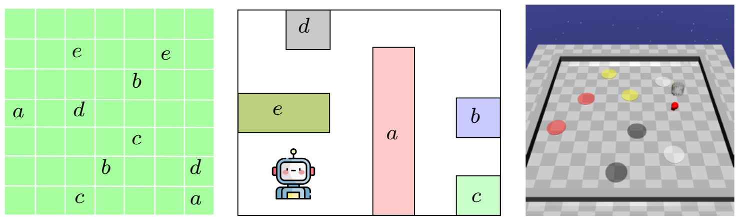
where λ is a hyper-parameter to balance two loss terms, chosen as 0.01 in this work.

Note that since V^h_θ is trained with R_i in the value loss of PPO, where R_i is a discounted accumulated rewards in multiple steps and hence V^h_θ is essentially updated by a multi-step Bellman operator (BO) [32].

4 Experiments

Our experiments aim to evaluate the performance of a multi-task RL agent trained using the proposed framework, focusing on learning efficiency, optimality, and generalization. Specifically, the section on overall performance examines whether the proposed framework outperforms baselines in terms of optimality and learning efficiency when sub-task dependencies are present. Next, ablation studies in Appendix investigate the impact of considering future sub-tasks within the low-level module, the contribution of experience relabeling to learning efficiency and the effect of latent transition model.

Before presenting the experiment results, we will first introduce the environments. Finally, the experiments about overall performance comparison will be demonstrated. The training setup and baselines are introduced in Appendix. Other experiment results and algorithmic details are deferred to Appendix.



(a) Letter (b) Walk (c) Zone

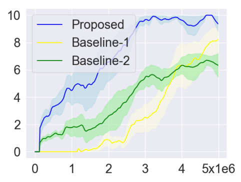
Fig. 6: Environments. The details of environments are introduced in Appendix.

4.1 Environments and Setup

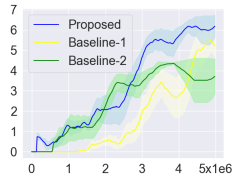
We conducted experiments in various environments featuring both discrete and continuous action and state spaces. Each environment is procedurally generated, with object layouts and positions randomized upon reset. The agent does not know the positions or properties of objects in advance, making it impossible to solve these environments using simple tabular-based methods. Each task is defined by a SPECTRL specification, expressed through symbolic propositions given by the labeling function. The agent's goal is to complete the specified task while maximizing accumulated rewards. The example screen shots of environments are shown in Figure 6. The detailed introduction of environments are in Appendix.

Sub-task Dependencies. In our experiments, sub-task dependencies can arise from various factors, including avoidance requirements, aliasing states, and subgoal reward functions. First, successfully completing one sub-task may necessitate avoiding the subgoals of another, thereby introducing dependencies between them. (Note that here, "sub-task" is defined by (2), which may not be explicitly represented in the task DAG.) Second, multiple states in the environment may be mapped to a single symbol through the labeling function, leading to aliasing states (e.g., multiple cells labeled with the same letter in Figure 6(a) and example in Figure 1). Because the agent must choose which state to visit conditioned on a particular subgoal symbol, these aliasing states cause the low-level policy of one sub-task to depend on future sub-tasks, thereby creating sub-task dependencies. Third, subgoal reward functions can induce dependencies if certain task specifications are more favorable to the user; for instance, in the Walk domain depicted in Figure 6(b), visiting "b" before "c" may yield a higher reward than going directly to "c," correlating the "b" and "c" sub-tasks. Additional experiments illustrating a wider range of sub-task dependencies are provided in the Appendix.

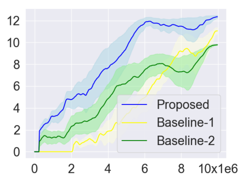
Note that the sub-task dependencies are unknown to the agent initially and the agent has to learn to adapt to these dependencies via interacting with the environment. The



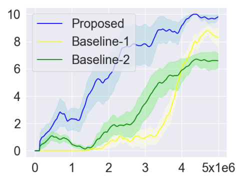
(a) Letter, First Set



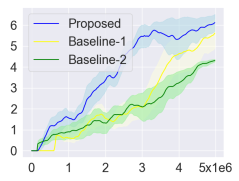
(b) Walk, First Set



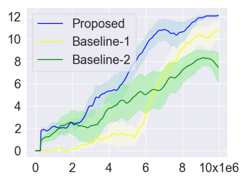
(c) Zone, First Set



(d) Letter, Second Set



(e) Walk, Second Set



(f) Zone, Second Set

Fig. 7: Performance Comparisons. The x-axis is the environmental step, and the y-axis is the average episodic return. In the first set of experiments, the reward is only given at the completion of the given task. In the second set, the reward of achieving a subgoal is dependent other subgoals achieved previously.

first two types of dependencies are at the low level, while the third type operates at a higher (subgoal) level.

4.2 Overall Performance

In this section, we present the overall performance comparisons in terms of average return for completing testing tasks. Other experiment results are included in Appendix. In each plot, the evaluation is conducted in every 10K training samples drawn from the environment. In each evaluation, 10 testing tasks are randomly generated and the average return of these tasks is used as the y-axis of the plot. The details of task generation for both training and testing are presented in Appendix.

In Figure 7, the proposed method is compared with two baselines introduced in Appendix. In the first set of experiments, the trained agent is evaluated on completing testing tasks with the minimum number of steps. Here, the reward is only provided upon completion of the task, and the agent must avoid any propositions specified in the safety condition. This set of experiments focuses on low-level sub-task dependencies, specifically requiring the agent to avoid subgoals of other sub-tasks and to reach designated subgoals while accounting for future subgoals. In the second set of experiments, in addition to task completion, rewards are also granted for achieving some subgoals under ordering constraints. Specifically, the reward for certain subgoals depends on previously achieved subgoals within the same episode, introducing sequential sub-task dependencies. Detailed information on rewards of sub-task dependencies is provided in the Appendix. In addition, we conduct the third set of experiments with parallel sub-task dependencies, which is presented in Appendix.

In Figure 7, we observe that the proposed framework outperforms Baseline-1 (LTL2Action [35]) in terms of learning efficiency. Unlike the proposed method, Baseline-1 relies solely on a task-conditioned policy and does not decompose the task to leverage its compositional structure, which hinders its learning efficiency. The faster convergence of the proposed framework results from the multi-step return estimation performed by the high-level module, which backpropagates rewards for each sub-task and provides return estimates to the low-level module upon the completion of each sub-task. In contrast, Baseline-1 propagates rewards step-by-step, resulting in slower learning convergence of its value function.

As shown in Figure 7, the proposed method outperforms Baseline-2 (Logic Option Framework [2]) in terms of both average return and learning efficiency. In Baseline-2, the low-level policy is specifically trained for each assigned sub-task, aiming to reach positive propositions and avoid negative propositions relevant to that sub-task, without regard to other sub-tasks. This lack of awareness of dependencies compromises global optimality in low-level behavior. Additionally, when subgoal rewards depend on one another, Baseline-2's high-level planning can only myopically select next sub-tasks to complete the rest of the task. This is because conventional planning methods, which is value iteration (VI) [2] in Baseline-2, cannot handle non-Markovian rewards of sub-tasks. These limitations explain Baseline-2's lower average return in both first and second sets of experiments.

5 Related Work

Applying the RL paradigm to solve logic compositional tasks has been explored in numerous prior studies. These methods typically start by converting the compositional task formula into an equivalent automaton representation and then create a product MDP by combining the environmental MDP with the task automaton [37]. Prominent approaches utilizing product MDPs include Q-learning for reward machines (Q-RM) [5, 11, 12], LPOPL [34], and geometric LTL (G-LTL) [23]. Additionally, [14] introduced the DiRL framework, which uses hierarchical RL to accomplish LTL tasks by integrating graph-based planning on the automaton to guide exploration for task satisfaction. However, these approaches are tailored to specific task formulas, requiring policies to be retrained from scratch for each new task. Thus, they lack zero-shot generalization.

Previous approaches have sought to train reusable skills or options to facilitate generalization in compositional task settings [1, 2, 20, 21]. In these methods, agents fulfill unseen tasks by sequentially combining pre-trained option policies through value iteration over potential subgoal choices, achieving satisfying success rate. However, they overlook inter-dependencies between subgoals, which can lead to suboptimal solutions when sub-tasks are dependent on each other, as demonstrated in Figures 1 and 2. Although [19] considers causal dependencies among sub-tasks, it requires these dependencies to be explicitly provided. In contrast, our framework does not require the agent to know sub-task dependencies beforehand, making it applicable to scenarios with general and implicit sub-task dependencies.

In [17, 35, 40], the authors propose task-conditioned policies to enable zero-shot generalization in compositional tasks by conditioning the policy on task embeddings extracted through recurrent graph neural networks [17] or graph neural networks [35, 40]. These methods can learn optimal policies for generalization with sufficient training. However, they did not address sparse reward and long horizon issues of DFA tasks specifically, so their learning efficiency was still not satisfying. This arises because they did not decompose tasks into sub-tasks or leverage the inherent compositional structure of these tasks. In this work, we introduce a hierarchical RL framework for zero-shot generalization that decomposes DFA task into reach-avoid sub-tasks and train the RL agent to generalize across different sub-tasks, addressing the issues of sparse reward and long horizon specifically.

In addition, authors in [18] proposed to compute successor features of propositional symbols as policy bias to guide the generalization of unseen LTL tasks. However, the successor features are only about the achievement of next symbols and do not encode any dependencies of sub-tasks to be completed in the future. So, their approach still ignores the sub-task dependencies and cannot achieve the optimality in zero-shot generalization.

Goal-conditioned reinforcement learning (GCRL) has long focused on training a unified policy for reaching arbitrary single goals within a specified goal space [25]. However, GCRL typically addresses scenarios where agents need to reach only a single goal per episode. In contrast, the compositional tasks in our work require achieving multiple subgoals under specific temporal order constraints. While some GCRL approaches introduce hierarchical frameworks that generate multiple subgoals within an episode [22, 6], these frameworks primarily aid exploration, with subgoals achieved in any order. This lack of temporal constraints makes these GCRL methods incompatible with our setting, so that these methods are not compared against the proposed framework.

There is a recent work [39] proposing to use future-dependent options to improve the optimality and learning efficiency of the generalization of compositional tasks. It is a similar work, but it does not consider safety issue and avoiding negative propositions of sub-tasks. This work proposes a new architecture and trains the agent to avoid negative propositions for the safety guarantee.

6 Conclusion

In this work, we propose a new hierarchical framework for generalizing compositional tasks in the SPECTRL language to address the sub-task dependencies and sparse reward issue. In the high level, we propose to use an implicit planner to select next sub-task optimally for the low-level module to complete. In the low level, the module is trained to complete the assigned sub-task, conditioned on the remaining task to complete. By transforming the given task into a DAG of reach-avoid sub-tasks, the sub-task dependencies are considered in the sub-task selection via an embedding of future sub-tasks extracted by a GNN, and the spare reward issue is resolved by the estimated return passed to the low-level module upon the completion of every sub-task. With comprehensive experiments, we demonstrate the advantages of the proposed framework over baselines in terms of optimality and learning efficiency. In the future, we plan to investigate the generalization of tasks specified by probabilistic logic language.

7 Acknowledgment

This work is supported by ARO grant under Award #W911NF-23-1-0146 and a gift by Open Philanthropy.

Bibliography

[1] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

[2] Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan DeCastro, Micah Fry, and Daniela Rus. The logical options framework. In *International Conference on Machine Learning*, pages 307–317. PMLR, 2021.

[3] Brandon Araki, Kiran Vodrahalli, Thomas Leech, Cristian-Ioan Vasile, Mark D Donahue, and Daniela L Rus. Learning to plan with logical automata. 2019.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[5] 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 *IJCAI*, 2019.

[6] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pages 1430–1440. PMLR, 2021.

[7] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.

[8] Floris den Hengst, Vincent François-Lavet, Mark Hoogendoorn, and Frank van Harmelen. Reinforcement learning with option machines. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2909–2915. International Joint Conferences on Artificial Intelligence Organization, 2022.

[9] Dhaval Gujarathi and Indranil Saha. Mt*: Multi-robot path planning for temporal logic specifications. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13692–13699. IEEE, 2022.

[10] Keliang He, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. Towards manipulation planning with temporal logic specifications. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 346–352. IEEE, 2015.

[11] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[12] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[13] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039, 2021.

[15] Danish Khalidi, Dhaval Gujarathi, and Indranil Saha. T: A heuristic search based path planning algorithm for temporal logic specifications. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8476–8482. IEEE, 2020.

[16] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020.

[17] 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. IEEE, 2020.

[18] David Kuric, Guillermo Infante, Vicenç Gómez, Anders Jonsson, and Herke van Hoof. Planning with a learned policy basis to optimally solve complex tasks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 333–341, 2024.

[19] Jonas Kvarnström and Patrick Doherty. Talplanner: A temporal logic based forward chaining planner. *Annals of mathematics and Artificial Intelligence*, 30:119–169, 2000.

[20] Borja G León, Murray Shanahan, and Francesco Belardinelli. Systematic generalisation through task temporal logic and deep reinforcement learning. *arXiv preprint arXiv:2006.08767*, 2020.

[21] 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*, 2021.

[22] Siyuan Li, Jin Zhang, Jianhao Wang, Yang Yu, and Chongjie Zhang. Active hierarchical exploration with stable subgoal representation learning. *arXiv preprint arXiv:2105.14750*, 2021.

[23] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*, 2017.

[24] Jason Xinyu Liu, Ankit Shah, Eric Rosen, George Konidaris, and Stefanie Tellex. Skill transfer for temporally-extended task specifications. *arXiv preprint arXiv:2206.05096*, 2022.

[25] Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

[26] V Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[28] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[29] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

[30] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[33] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[34] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018.

[35] Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.

[36] Elise van der Pol, Thomas Kipf, FA Oliehoek, and Max Welling. Plannable approximations to mdp homomorphisms: Equivariance under actions. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2020*, volume 2020. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2020.

[37] Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In *International Conference on Machine Learning*, pages 35137–35150. PMLR, 2023.

[38] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5064–5078, 2022.

[39] Duo Xu and Faramarz Fekri. Generalization of temporal logic tasks via future dependent options. *Machine Learning*, pages 1–32, 2024.

[40] Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit Seshia. Compositional automata embeddings for goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 37:72933–72963, 2024.

[41] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.