Generalization of Temporal Logic Tasks via Future Dependent Options

Duo Xu and Faramarz Fekri

August 28, 2024

Generalization of Temporal Logic Tasks via Future Dependent Options

Duo Xu[1*] and Faramarz Fekri[1]

[1] Department of Electrical and Computer Engineering,
Georgia Institute of Technology,
5th North Ave, Atlanta, 30332, GA, USA.

*Corresponding author(s). E-mail(s): dxu301@gatech.edu;

Abstract

Temporal logic (TL) tasks consist of complex and temporally extended subgoals and they are common for many real-world applications, such as service and navigation robots. However, it is often inefficient or even infeasible to train reinforcement learning (RL) agents to solve multiple TL tasks, since rewards are sparse and non-Markovian in these tasks. A promising solution to this problem is to learn task-conditioned policies which can zero-shot generalize to new TL tasks without further training. However, influenced by some practical issues, such as issues of lossy symbolic observation and long time-horizon of completing TL task, previous works suffer from sample inefficiency in training and sub-optimality (or even infeasibility) in task execution. In order to tackle these issues, this paper proposes an option-based framework to generalize TL tasks, consisting of option training and task execution parts. We have innovations in both parts. In option training, we propose to learn options dependent on the future subgoals via a novel approach. Additionally, we propose to train a multi-step value function which can propagate the rewards of satisfying future subgoals more efficiently in long-horizon tasks. In task execution, in order to ensure the optimality and safety, we propose a model-free MPC planner for option selection, circumventing the learning of a transition model which is required by previous MPC planners. In experiments on three different domains, we evaluate the generalization capability of the agent trained by the proposed method, showing its significant advantage over previous methods. The source code is available at https://drive.google.com/drive/folders/128jGb3HkPbXPj-FO6FQ7NW3LTFkqUUQy?usp=drive_link

Keywords: Temporal logic task, reinforcement learning, neuro-symbolic, option learning

1 Introduction

Reinforcement learning (RL) is a promising framework for developing truly general agents capable of acting autonomously in the real world, ranging from video games [1, 2] to robotics [3, 4]. Previous RL algorithms primarily focus on solving tasks with a single goal state. However, many real-world applications may require agents to satisfy temporally extended goals (e.g., eventually take the key and then reach the door). Tasks consisting of temporally extended subgoals are termed as temporal logic (TL) tasks [5, 6]. TL tasks have a wide range of applications in the real world, such as control system and robotics. For example, a service robot on the factory floor might have to fetch the a set of components but in different orders depending on the product being assembled, and it may need to avoid some unsafe situations. However, since reward function is sparse and non-Markovian in TL tasks, most previous RL algorithms may have poor performance and low learning efficiency.

Generalization to multiple TL tasks is a key requirement for deploying autonomous agents in many real-world domains [7]. It is important for RL agent to learn to perform zero-shot execution of different tasks by leveraging the generalization abilities of deep learning models. However, previous related works suffer from various deficiencies [8–12], such as optimality and learning efficiency. Some works [9, 11, 12] solve new TL tasks by leveraging the learned reusable skills or options, but produce sub-optimal or even infeasible solutions, especially when the symbolic observation is lossy, i.e., one symbolic state can correspond to multiple environment states. These methods train an independent option for reaching a specific subgoal specified in terms of the environmental propositions, as we illustrated in an example in Section 4, which may fail to achieve the global optimality of the task completion in the case of lossy symbolic observation. Further, [8, 10] propose to train policies conditioned on the task specification formula directly, where the agent needs a large amount of environment samples to learn to understand temporal operators and figure out the optimal path for satisfying the task. These approaches have poor sample efficiency in complex tasks or environments, since they do not utilize reusable skills when solving compositional tasks like TL tasks.

In this work, in order to tackle the above issues, we propose a hierarchical option-based framework to solve the TL tasks and generalize to any new task without further learning. The proposed framework consists of option training part and task execution part.

In the *option training* part, we train a generalizable agent which can achieve various subgoals with different future situations. We propose two innovations for training options: 1) We introduce future-dependent options which are trained to not only achieve the target subgoal, but also consider other subgoals to be achieved in the future. By preventing myopic behavior in achieving the target subgoal, the proposed options can approximate the global optimality of the task completion as much as possible. 2) Since the task of achieving a sequence of temporally extended subgoals may have long time horizon, learning policies of future dependent options needs reward information of many future time steps ahead. In order to facilitate the reward propagation in this case, we train a multi-step value function to predict the discounted return of satisfying future subgoal sequence.

In the *task execution* part, the given TL task is solved by a hierarchical option framework, where the high level is for option selection and the low level is for option execution. In the high level, whenever the previously selected option is successfully finished, the proposed model-free option planner first finds the optimal sequence of subgoals which not only satisfies the given TL task and but also has the largest expected return in the multi-step value function, and then selects the option for reaching the first subgoal conditioned on other subgoals in the optimal sequence. This option planner works in a manner of model predictive control (MPC) [13], but it does not need to learn a transition model and hence circumvents the compounding errors caused by the inaccuracy of the learned transition model during planning. The advantage of the proposed planner is also theoretically justified.

In experiments, we demonstrate the zero-shot generalization capability of the trained agent in three environments, including both discrete and continuous action spaces. All these environments are procedurally generated where the layout and task specification are randomly generated, so that none of tasks can be solved by simple tabular methods [14]. With comprehensive evaluations, we show that the proposed approach outperforms previous representative methods in terms of sample efficiency, accuracy and optimality.

2 Related Work

Extending the RL paradigm to solve multiple temporal logic tasks has been studied by many previous works. These approaches augment the state space and obtain an equivalent product MDP by transforming the TL task formula into its automaton equivalence. Representative previous approaches, such as Q-learning for reward machines (Q-RM) [15–17], LPOPL [6] and geometric LTL (G-LTL) [18], augment the environment state space with the automaton transformation of the LTL specification. In addition, authors in [19] proposed a DiRL framework to complete LTL task successfully by using hierarchical RL which interleaves graph-based planning on the automaton and guide the agent's exploration for task satisfaction. However, although the compositional nature of the TL task is utilized in these approaches, the compositionality is not leveraged in generalization to novel tasks. As such, the agent must learn the policy for satisfying a new TL task from scratch.

Learning independent option policies or skills for achieving each subgoal has been a common approach towards generalization in a TL task setting [9, 20–22]. For any unseen task specification, the agent sequentially composes these option policies to satisfy the task. However, these methods require a lot of additional fine-tuning to satisfy the task correctly and they cannot address the issue of lossy symbolic observation. Therefore, the optimality and even feasibility of the solution cannot be guaranteed. Instead, we propose a general framework for transferring learned policies to novel specifications in a zero-shot setting.

Authors in [8] proposed learning a modular policy network by composing subnetworks via a recurrent graph neural network for each proposition and operators, based on the syntax tree transformed from the TL task specification. Given a new task specification, the final policy network is created by composing the subnetwork modules in the new syntax tree corresponding to the given specification. The work in [10] proposes to use graph convolutional networks to learn an embedding for the given TL specification to tackle novel TL specifications. However, since the task specification is processed in its original form, the agent needs a lot of environmental interactions to learn temporal operators and figure out the optimal path to satisfy the task. These approaches may result in unsatisfactory performance on sample efficiency or optimality when the task specification has complex logic relationships. We compare these approaches with ours in experiments.

Agents learning to act in a partially observable domain may need to overcome the problem of perceptual aliasing, i.e., different states that have similar observations but require different responses. The lossy symbolic observation can be regarded as the perceptual aliasing in the symbolic domain, which is an important problem in POMDP studied for decades [23–26]. Our work tackles it in the option-level, distinguishing different options which have same terminal conditions (states) but require different option policies to reach terminal states. In contrast to previous approaches [25], our framework distinguish options with perceptually aliasing terminal states by reward information of reaching future subgoals, based on which different option policies are trained. In addition, there are a surge of recent papers studying goal-conditioned RL which train a unified policy to reach arbitrary goals [27]. However, these GCRL papers solved a different problem from satisfying TL tasks, since the agent in GCRL problem only needs to reach a single goal in each episode. Besides, some GCRL papers also proposed to generate subgoals in a hierarchical framework [28, 29]. But these subgoals are only used to facilitate the agent to reach distant goals, without any strict temporal orders or relationships. Therefore, GCRL papers are not comparable with our work.

3 Preliminaries

3.1 Reinforcement Learning

RL provides a framework for learning to select actions in an environment in order to maximize the collected rewards over time [14]. RL deals with problems formalized as Markov decision processes (MDP). We denote an environment MDP as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma, S_0 \rangle$, where $\mathcal{S}$ is a finite set of environment states, $\mathcal{A}$ is a finite set of agent actions, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function, $R : \mathcal{S} \times \mathcal{A} \to [R_{\text{min}}, R_{\text{max}}]$ is a reward function with $R_{\text{min}}, R_{\text{max}} \in \mathbb{R}$, $\gamma \in [0, 1)$ is a discount factor, and $S_0 : s_0 \sim S_0$ is a distribution of initial states. In each time step t, the agent observes the environment state s_t and selects an action a_t to apply, according to a policy function $\pi \in \Pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and then collects reward $r_t = R(s_t, a_t)$.

For some policy π, the values V and Q for any state s and state-action pair (s, a) at time t can be defined as below,

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | s_t = s \right] \tag{1}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | s_t = s, a_t = a \right], \tag{2}$$

where $\mathbb{E}_\pi$ is the expectation of accumulated rewards following some policy π. A policy is the optimal policy π^* if it produces the highest accumulated rewards: $\forall s \in \mathcal{S}, \forall \pi \in \Pi, \forall a \in \mathcal{A} : Q_{\pi^*}(s, a) > Q_\pi(s, a)$. Searching π^* can be addressed by parameterizing the policy and finding optimal parameters θ^* that maximize the accumulated rewards by a policy optimization algorithm. Specifically, parameters θ can be weights of neural networks optimized by gradient descent.

The policy optimization algorithms can be categorized into off-policy and on-policy methods [14]. In off-policy methods, such as SAC [30], the policy π is optimized by experience tuples (s_t, a_t, r_t, s_{t+1}) generated by old policies in previous iterations and stored in a replay buffer $\mathcal{B}$. The weights θ of Q function are updated by minimizing the loss function as below,

$$\mathcal{L}(\theta;\theta^-) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\left(r + \gamma \max_{a'} Q(s',a'|\theta^-) - Q(s,a|\theta)\right)^2 \tag{3}$$

where θ^- are target weights of neural networks which are updated periodically for improving numerical stability of the learning process [31]. In on-policy method, such as PPO [32], the policy π is optimized by experience tuples generated by current policy and stored as trajectories τ. The parameters ϑ of V function are updated by minimizing the loss as below,

$$\mathcal{L}(\vartheta;\vartheta^-) = \mathbb{E}_{(s,a,r,s')\sim\tau}\left(r + \gamma V(s'|\vartheta^-) - V(s|\vartheta)\right)^2 \tag{4}$$

where ϑ^- are target parameters copied periodically from ϑ for better numerical stability [31].

3.2 Option Framework

The option framework was introduced in [33] to incorporate temporally-extended actions (options) into reinforcement learning. An option $o = \langle \mathcal{I}, \beta, \pi \rangle$ is defined by three elements: 1) the initiation set $\mathcal{I}$ denotes the states where the option can be started to execute; 2) the termination condition β defines the condition when option execution ends; 3) the option policy π selects actions to take the agent to realize β starting from any state in $\mathcal{I}$. We leverage the options framework to generalize temporal logic tasks by re-usable skills. In our framework, an option is defined as the skill of achieving a specific subgoal.

3.3 Temporal Logic Task Specification

A temporal logic (TL) task is described by a TL specification φ, a Boolean function that determines whether the objective formula is satisfied by the given trajectory or not [34]. The specification of TL task is used to express (multi-task) temporally extended subgoals and partial orders of subgoals for task completion [35]. First define a common vocabulary $\mathcal{AT}$ as the set of atomic tasks. TL tasks are widely used in real-world applications. For instance, in service robot applications, $\mathcal{AT}$ could include events such as opening the drawer, activating the fan, turning on/off the stove, or entering the bathroom. Then, the TL task can include temporally-extended occurrences of these events. For example, two possible TL tasks are (1) "Open the drawer and activate the fan in any order, then turn on the stove" and (2) "Open the drawer but do not enter the bathroom until the stove is turned off".

In order to study the systematic generalization of TL tasks with options, we adopt task temporal logic (TTL) [22] to specify TL tasks. TTL is designed to be an expressive, learning-oriented TL language interpreted over finite traces, i.e, over finite episodes. The language of TTL is a fragment of the widely-used Linear-time Temporal Logic over finite traces (LTLf) [5], and the translation of any TTL formula into LTLf is provably guaranteed in [21]. TTL is expressive enough to represent tasks in [20] which is a popular benchmark in the RL-TL literature.

Definition 1 (Task Temporal Logic [22]). Given the vocabulary $\mathcal{AT}$, every formula φ in TTL is built from atomic tasks $a \in \mathcal{AT}$, negation $\neg$ (on proposition only), and sequential composition

";", connected by operators $\vee$, $\cup$, and $\mathcal{U}$. The grammar of TTL is expressed as below:

$$l ::= a|\neg a|l \vee l', \qquad \alpha ::= l\mathcal{U}l', \qquad \varphi ::= \alpha|\varphi;\varphi'|\varphi \cup \varphi' \tag{5}$$

where | indicates the alternative choices between templates. Specifically, an atomic task a represents the reachability of a corresponding subgoal in the environment. So, every atomic task a in TTL means that its associated subgoal a needs to be achieved eventually, i.e., $\Diamond a$ in LTL language [5]. The operators $\vee$ and $\cup$ represent the non-deterministic choice. The operator $\mathcal{U}$ refers to "until" and $l\mathcal{U}l'$ reads l *must hold before* l' *is satisfied*. Besides, the sequential composition $\varphi;\varphi'$ represents that the formula φ' has to become satisfied after φ holds true. The temporal operators *eventually* $\Diamond$ and *always* $\Box$ can be also defined by operator $\mathcal{U}$.
Definition 2 (Satisfaction). The truth value of a TL task specification is determined by a finite sequence of truth assignments $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_N \rangle$ with vocabulary $\mathcal{AT}$, where $a \in \sigma_i$ iff the atomic task a is achieved at time step i. Then, σ satisfies φ (5) at time $i \geq 0$, denoted by $\langle \sigma, i \rangle \models \varphi$:

- $\langle \sigma, i \rangle \models a$ iff $a \in \sigma_i$, where $a \in \mathcal{AT}$
- $\langle \sigma, i \rangle \models \neg a$ iff $\langle \sigma, i \rangle \not\models a \in \mathcal{AT}$
- $\langle \sigma, i \rangle \models (a \vee a')$ iff $\langle \sigma, i \rangle \models a$ or $\langle \sigma, i \rangle \models a'$
- $\langle \sigma, i \rangle \models l\mathcal{U}l'$ iff there exists j such that $i \leq j \leq N$ and $\langle \sigma, j \rangle \models l$, and $\langle \sigma, k \rangle \models l'$ for all $k \in [i, j)$
- $\langle \sigma, i \rangle \models \varphi;\varphi'$ iff there exists j such that $i < j \leq N$, and $\langle \sigma, i \rangle \models \varphi$, and $\langle \sigma, j \rangle \models \varphi'$
- $\langle \sigma, i \rangle \models (\varphi \cup \varphi')$ iff $\langle \sigma, i \rangle \models \varphi$ or $\langle \sigma, i \rangle \models \varphi'$

A sequence σ is defined to satisfy φ iff $\langle \sigma, 0 \rangle \models \varphi$. This sequence σ here is same as the sequence of symbolic observations in the rest of this paper.
Progression Technique. The progression function, denoted as $\text{prog}(\sigma, \varphi)$, is defined as a function which takes a TL specification and the current labelled state (symbolic observation) as inputs and returns a formula that expresses aspects of the original formula that remain to be satisfied [10, 36]. For example, in Figure 1, consider task $\varphi :=$ wood; diamond; ax, i.e., collect wood, then diamond and ax finally. It will be progressed to "diamond; ax" after wood is collected, meaning that the agent still needs to collect diamond and then ax.

3.4 Problem Formulation

We now formulate the problem of learning an RL agent that can zero-shot solve various TL tasks in a finite episode. Assume that the agent is working on an environment MDP $\mathcal{M}_e = \langle \mathcal{S}, \mathcal{A}, T, R_e, \gamma, S_0 \rangle$ equipped with a labelling function $L : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{P}}$, mapping a state-action pair into a set of propositions from $\mathcal{P}$. We assume the propositions in $\mathcal{M}_e$ consist of subgoal propositions $\mathcal{P}_G$ and events $\mathcal{P}_E$, i.e., $\mathcal{P} = \mathcal{P}_G \cup \mathcal{P}_E$. Assume that the set of subgoals $\mathcal{G}$ is specified by the user and known as a characteristic of the environment. Each option is trained to achieve a subgoal $g \in \mathcal{G}$ and some subgoals may consist of multiple subgoal propositions from $\mathcal{P}_G$, i.e., any subgoal task $g \in \mathcal{G} \subset 2^{\mathcal{P}_G}$. We use $\wedge$ as conjunction of propositions holding true at the same time. For example, assuming that $\mathcal{P}_G = \{a, b, c\}$, the subgoals can be $\mathcal{G} = \{a \wedge b, c, c \wedge d\}$ and $a \wedge b$ means propositions a and b hold true at the same time. The *subgoal sequence* used in this work is a sequence of subgoals from $\mathcal{G}$, and we also use $\mathcal{G}$ as atomic tasks $\mathcal{AT}$ for formulating TTL specifications φ defined in (5).

In this paper, the proposed framework aims at solving the problem of maximizing returns in a multi-task MDP which is a product of environment MDP and TL tasks, defined as below.

Definition 3. (Multi-task MDP) Given an environment MDP $\mathcal{M}_e = \langle \mathcal{S}, \mathcal{A}, T, R_e, \gamma, S_0 \rangle$ defined in Section 3.1, a finite set of propositions $\mathcal{P}$, a labelling function $L : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{P}}$, a finite set of TL specifications Φ, and a probability distribution τ over Φ, we construct a multi-task MDP as $\mathcal{M}_\Phi = \langle \mathcal{S}', \mathcal{A}, T', R', S'_0 \rangle$, where $\mathcal{S}' = \mathcal{S} \times \Phi$, $T'(s', \varphi' | s, \varphi, a) = T(s'|s, a)$ only when $\varphi' = \text{prog}(L(s, a), \varphi)$ (zero otherwise), $S'_0(s, \varphi) = S_0(s) \cdot \tau(\varphi)$, and

$$R'(\langle s, \varphi \rangle, a) = \begin{cases} R_F & \text{if } \text{prog}(L(s,a), \varphi) = \text{true} \\ -R_F & \text{if } \text{prog}(L(s,a), \varphi) = \text{false} \\ R_e & \text{otherwise} \end{cases}$$

where R_F is empirically selected, and S'_0 and S_0 are distributions of initial states in the environment and multi-task MDPs, respectively.

4 Methodology

In this work, we propose an option-based framework to solve and generalize tasks with TL specifications. Here every option is trained to achieve a specific subgoal conditioned on future ones. This is motivated by two important issues here. The first issue is *the lossy symbolic observation*, meaning that a single propositional symbol can correspond to multiple different environment states, as explained in the following motivating example.

The second issue is that, the long horizon of satisfying the temporally extended subgoal sequence can make rewards difficult to propagate throughout the state space. In order to address these issues, we propose two innovations for option training, future-dependent option and multi-step value function. For task execution, by leveraging these two innovations we propose a novel model-free planner for option selection, whose advantage is also theoretically justified.

In the following, we first present a motivating example. Then the general framework of option training and task execution is illustrated. Following that, we present future dependent option and multi-step value function with details. Finally, the algorithms, especially the option planner, are introduced.

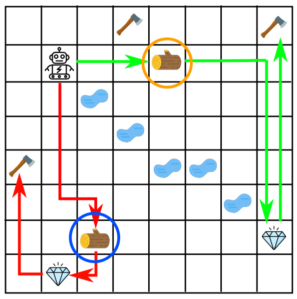
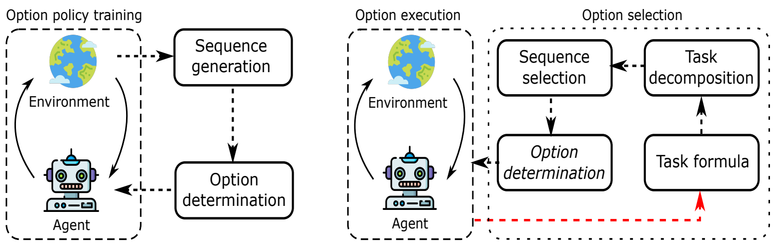


Figure 1: Motivating example. The TL task is wood; diamond; ax (go to collect wood, then diamond and finally ax). The state s_A denotes the state when the agent is at the wood in the orange circle, and s_B denotes that the agent is in the blue circle.

4.1 Motivation

In Figure 1, assume that environment reward R_e is -0.1 for every movement and the given task is $\varphi :=$ wood; diamond; ax (go to collect wood, then diamond and finally ax). There are two choices (s_A and s_B) for the agent to collect wood. Previous option-based approaches may myopically choose to collect the wood in the orange circle since it is closer, and finish the task φ along the green path. However, considering R_e, the globally optimal solution of task φ is the red path. In some cases, the decision made by myopic option-based approaches may lead to infeasible solutions. For instance, if the game



(a) Option Training (b) Task Execution

Figure 2: The proposed framework of option training and task execution. In task execution, the option selection part is a *model-free option planner*. The red arrow indicates that the task formula is progressed by the subgoal symbol g when g is achieved.

in Figure 1 has constraint that the agent cannot move more than 12 steps in one episode, the green path with myopic choice of collecting wood is infeasible.

In this work, to address issues mentioned above, we propose a novel option framework where every option is dependent on a sequence of future subgoals. We denote $\mathcal{O}$ as the set of options. Let o_g^ξ denote the option of reaching subgoal $g \in \mathcal{G}$ conditioned on ξ as a sequence of future subgoals to satisfy. We train each option $o_g^\xi \in \mathcal{O}$ not only by the experience of reaching the subgoal g, but also with the reward information of satisfying subgoals in ξ (in a fixed order same as ξ), since the option o_g^ξ is to reach subgoal g but also conditioned reaching the subgoal sequence ξ. Therefore, in order to back-propagate the reward information of satisfying subgoal sequence ξ (which often has long time horizon to complete), we also train a multi-step value function $V^\phi(s;\xi)$ to predict the discounted return obtained by reaching subgoals in ξ starting from the state s. We use V^ϕ to set target values to update value functions of options, hence accelerating the training efficiency of options.

Consider the example in Fig. 1 again, when the option of collecting "wood" is also trained with reward information of collecting diamond and then ax after collecting wood, i.e., $g =$"wood" and $\xi :=$["diamond", "ax"], the option policy π_g^ξ will lead the agent to s_B (the wood in blue circle) instead of s_A. This is because $V^\phi(s_A;\xi) < V^\phi(s_B;\xi)$ and V^ϕ sets the targets for updating value function of option policy π_g^ξ.

4.2 General Framework

The proposed framework is presented in Figure 2. The option training part is to learn the policies of future-dependent options determined by subgoal sequences with various compositions and lengths. These learned options lay the foundation of agent's capability of generalization to new TL tasks. Every training iteration works as below. The detailed training algorithm is presented in Algorithm 1 in Appendix A.7.

1. *Sequence generation:* the environment randomly generates subgoal sequences τ which only consist of subgoals from $\mathcal{G}$. For example, with $\mathcal{G} = \{a, b, c, d, e\}$, the subgoal sequence $\tau = [a, b, d]$ asks the agent to first reach a, then b and finally d, where a, b, d are subgoals;
2. *Option determination:* given the subgoal sequence τ, the agent needs to determine appropriate future dependent options to achieve subgoals in τ one-by-one. For example, if

$\tau = [a, b, d]$, the options determined to be trained should be o_a^{bd}, o_b^d and $o_d^{\varnothing}$ ($\varnothing$ means empty);

3. *Option policy training:* the policies of determined options are trained by using an appropriate RL algorithm together with agent's experience of trying to satisfy subgoals in τ. During the training, the reward propagation is augmented by the multi-step value function $V^{\phi}(\cdot; \tau)$ which is introduced in Section 4.4;

The task execution part in Figure 2(b) is a hierarchical framework for solving given TL tasks without further training based on learned options, where the option selection is conducted in the high level and the selected option is executed in the low level. In the high level, the option selection is realized by a *model-free option planner*, where the task formula is first decomposed into the set of satisfying subgoal sequences, then the optimal subgoal sequence is selected and finally the optimal option to be executed is determined. In the low level, the policy of selected option is executed in the environment. More details about option planner are introduced in Section 4.5.1.

1. *Task decomposition:* The given task φ is first decomposed into a set $\mathcal{K}$ of subgoal sequences τ_i, i.e., $\mathcal{K} := \{\tau_i\}_{i=1}^{M_\varphi} = \{[g_1^i, g_2^i, \ldots, g_{L_i}^i]\}_{i=1}^{M_\varphi}$, where each τ_i can satisfy φ and any g_j^i is from $\mathcal{G}$. For instance, if $\varphi = a; (b \vee c); (d \vee e)$, the decomposed subgoal sequences are $\mathcal{K} = \{[a, b, d], [a, b, e], [a, c, d], [a, c, e]\}$ and $M_\varphi = 4$. The details are in Algorithm 3 in Appendix;
2. *Sequence selection:* The optimal subgoal sequence τ_{i^*} with highest expected return is selected from $\mathcal{K}$ according to the multi-step value function V^{ϕ}, i.e., $i^* = \arg\max_{i \in [1, M_\varphi]} V^{\phi}(s_0; \tau_i)$ and s_0 is the initial state;
3. *Option determination:* Different from that in the option training part, only the *first* option from τ_{i^*}, i.e., $o_{\tau_{i^*}[0]}^{\tau_{i^*}[1:]}$, is determined and sent to the low level for execution. For example, assuming $\tau_{i^*} = [a, b, d]$, only o_a^{bd} is selected and executed in the low level;
4. *Option policy execution:* The agent executes the policy of option selected above in the environment. When the selected option is successfully finished, the task formula φ is updated by the progression with the achieved subgoal. For example, assuming that the target task is $\varphi = a; (b \vee c); (d \vee e)$ and the execution of option o_a^{bd} is successfully finished, the task formula φ will be progressed by the subgoal symbol a and become $\varphi := \text{prog}(\varphi, a) = (b \vee c); (d \vee e)$.

Remark. The future-dependent property makes the option selection non-Markovian, and this is reason for performance improvement. In previous option frameworks, such as [21, 22], every option is trained to reach a specific subgoal, and option selection is independent of other options and hence Markovian. However, this is shown to be sub-optimal in the motivating example of Section 4.1, where making options dependent on the future can produce globally optimal solution. However, future-dependent option is non-Markovian, since its policy is dependent on future subgoals.

4.3 Future Dependent Option

In this work, we define a future dependent option, i.e., $o_g^{\xi} := \langle \mathcal{S}, \beta_g, \pi_g^{\xi} \rangle$ where ξ is a finite sequence of subgoals to be achieved in the future after completing g. Each option is trained to achieve its corresponding subgoal $g \in \mathcal{G}$ conditioned on future subgoals in ξ, working in the multi-task MDP (Definition 3 in Section 3.4). For each option, without loss of generality, the initial set is the same as the state space $\mathcal{S}$, and the terminal function is the indicator of satisfying the subgoal g, i.e., $\beta_g(s) = \mathbf{1}\{L(s) \models g\}$ where $L(\cdot)$ is the labeling function defined in Section 3.3. The option policy π_g^ξ is trained to maximize the discounted return of achieving the target subgoal g conditioned on achieving the future subgoals in sequence ξ, encouraging the option policy to realize the global optimality of achieving both g and ξ. Specifically, since achieving the future subgoal sequence ξ is taken into consideration, the return of achieving ξ needs to be back-propagated to train the action value (Q) function of the option π_p^ξ, which is helped by the multi-step value function V^ϕ introduced in next section.

Generally, when the option policy is trained by an off-policy method, the agent learns a sample-based approximation to the Q function $Q_{\pi_g^\xi}(s, a)$ of option o_g^ξ in (2), denoted as $Q_g^\theta(s, a; \xi)$ referring to the expected discounted return of achieving subgoal g conditioned on the achievement of subgoal sequence ξ in the future. Alternatively, when the option policy is trained by an on-policy method, the agent learns a value function $V_g^\theta(s; \xi)$ to approximate the value function $V_{\pi_g^\xi}(s)$ of option o_g^ξ in (1). The Q (or V) function of option is updated by TD-1 method as (3) (or (4)). For different environments, we choose an appropriate RL method to learn option policies, such as SAC for off-policy or PPO for on-policy method.

In option training, the option policy π_g^ξ is trained together with other options used to satisfy future subgoals in ξ. Specifically, given any subgoal sequence $\xi := \{g_i\}_{i=1}^K$ in option training, we define the sub-sequences $\xi_k := \{g_i\}_{i=k+1}^K$ and $k = 1, \ldots, K$. We then start from trying the option policy $\pi_{g_1}^{\xi_1}$ to achieve subgoal g_1. When the subgoal g_k in ξ is satisfied, we switch to use another option policy $\pi_{g_{k+1}}^{\xi_{k+1}}$ to achieve g_{k+1}, repeating this process until the agent satisfies the last subgoal g_K by using the option policy $\pi_{g_K}^{\varnothing}$. In addition to environmental rewards, the agent will receive the reward R_F (defined in Section 3.4) when the last subgoal g_K is satisfied. For any $k = 1, \ldots, K-1$, the discounted returns during the executions of option policies from $\pi_{g_{k+1}}^{\xi_{k+1}}$ to $\pi_{g_K}^{\varnothing}$ are all back-propagated to train the policy $\pi_{g_k}^{\xi_k}$ (updating $Q_g^\theta(\cdot, \cdot; \xi_k)$ or $V_g^\theta(\cdot; \xi_k)$), via the multi-step value function V^ϕ introduced in Section 4.4.

It is worth noting the difference between V_g^θ and V^ϕ. The value function $V_g^\theta(\cdot; \xi)$ is used to train option policy π_g^ξ if on-policy training method is used, and it is associated with the option o_g^ξ and subgoal g. However, $V^\phi(\cdot; \xi)$ is the multi-step value function used to propagate reward information of satisfying the subgoal sequence ξ, independent of any options or subgoals.

4.4 Multi-step Value Function

Since the value functions of option policies (Q_g^θ or V_g^θ) are updated with TD-1 method in (3) or (4), each update can propagate the reward information for only one time step. However, we note that the satisfaction of a future subgoal sequence ξ can have long horizon with sparse rewards, and the training of option policy π_g^ξ is dependent on satisfying ξ. Therefore, it can be inefficient to propagate the reward information of satisfying ξ back to update Q_g^θ or V_g^θ in training the option policy π_g^ξ. In the rest of the paper, we use $\xi[k]$ to denote the k-th subgoal in sequence ξ.

In order to help the propagation of reward information in long-horizon tasks, we propose to learn a multi-step value function $V^\phi(s; \xi)$ to estimate the discounted return of satisfying the subgoal sequence ξ starting from state s. In option training, the output of V^ϕ is used to

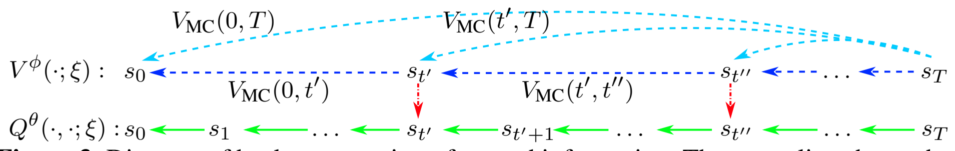


Figure 3: Diagram of back-propagation of reward information. The green line shows that Q functions of different options (target subgoals are omitted) are learned by TD-1 method. Note that function V^{ϕ} is independent of subgoals while Q^{θ} is dependent on different subgoals. The first and second subgoals ($\xi[0]$ and $\xi[1]$) are satisfied at $s_{t'}$ and $s_{t''}$, respectively. The red line shows that the multi-step value function V^{ϕ} sets the target value for Q^{θ} whenever a subgoal is satisfied. The blue and cyan curves denote the Monte Carlo estimate of multi-step discounted return $V_{\text{MC}}(\cdot,\cdot)$. V^{ϕ} is updated whenever a subgoal is satisfied, which has much coarser time resolution than Q^{θ}.

set the target value for updating $Q^{\theta}_{g}(\cdot,\cdot;\xi)$ or $V^{\theta}_{g}(\cdot;\xi)$ so that the reward propagation toward option policies can be accelerated. In Figure 3, it visually shows how the reward information is back-propagated in both Q^{θ} and V^{ϕ} when options are learned by an off-policy RL method. In the on-policy case, V^{θ} and V^{ϕ} work in the same way. Additionally, this function $V^{\phi}(\cdot;\xi)$ is also used to build a model-free option planner in task execution.

Specifically, the target value for updating V^{ϕ} is calculated based on Monte Carlo (MC) estimates of two discounted returns. 1) The first is the MC estimate of the discounted return till the end of the trajectory (cyan curves in Figure 3), i.e., $V_{\text{MC}}(t,T) := \sum_{k=t}^{T}\gamma^{k-t}r_t$ (T is the last time step of the trajectory). We use $V_{\text{MC}}(t,T)$ here since it is unbiased and good at capturing long-term rewards, but it also has large variance [14]; 2) In order to attenuate the variance, we also use the MC estimate of the discounted return till the satisfaction of next subgoal $\xi[0]$ (blue curves in Figure 3), i.e., $V_{\text{MC}}(t,t') = \sum_{k=t}^{t'}\gamma^{k-t}r_t$ (t' is the time when $\xi[0]$ is satisfied). This $V_{\text{MC}}(t,t')$ is used to build a multi-step temporal difference (TD) target for updating V^{ϕ}, which is together with the value estimate of satisfying other subgoals $\xi[1:]$ from a lagged value network V^{ϕ^-} [31].

Assume we have a trajectory $\tau = \{s_0, a_0, r_0, s_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$ with the subgoal sequence ξ as the target task to complete. If next subgoal $\xi[0]$ is satisfied at time t', the target for multi-step V^{ϕ} function is written as

$$V^{\text{target}}(s_t;\xi) = \max\{V_{\text{MC}}(t,t') + \gamma^{t'-t}V^{\phi^-}(s_{t'};\xi[1:]), V_{\text{MC}}(t,T)\} \tag{6}$$

In the equation above, the first term in the maximum is a multi-step TD target formed by $V_{\text{MC}}(t,t')$ together with the lagged value network V^{ϕ^-}. As discussed above, we also use $V_{\text{MC}}(t,T)$ in (6) to directly capture the reward information till the end of the trajectory. Since the value network always has very small values throughout the state space in early training stages, we need to use a maximum operator in (6) to help the reward back-propagate from the end of the trajectory. If next subgoal $\xi[0]$ is not satisfied by any state in τ, the target will become $V^{\text{target}}(s_t;\xi) = \max\{V^{\phi^-}(s_t;\xi), V_{\text{MC}}(t,T)\}$. Finally, the value function V^{ϕ} is trained to predict its target value by minimizing the loss function

$$J(\phi) = \ell(V^{\phi}(s_t;\xi), V^{\text{target}}(s_t;\xi)) \tag{7}$$

where ℓ is an arbitrary differentiable loss function.

The value function V^ϕ also sets the target value to update the Q functions of option policies (i.e., $Q_g^\theta(\cdot,\cdot;\xi)$) whenever the subgoal g is satisfied. For any tuple (s_t, a_t, r_t, s_{t+1}), the target value for $Q_g^\theta(\cdot,\cdot;\xi)$ is expressed as,

$$\begin{aligned} Q_g^{\text{target}}(s_t, a_t, r_t, s_{t+1};\xi) &= r_t + \gamma\beta_g(s_{t+1})V^\phi(s_{t+1};\xi) \\ &+\gamma(1-\beta_g(s_{t+1}))\max_{a'} Q_g^{\theta^-}(s_{t+1}, a';\xi) \end{aligned} \tag{8}$$

where θ^- is the parameter of the lagged target network as [31]. This target means that when g is not satisfied yet (i.e., $\beta_g(s_{t+1})$ =false), the Q function is updated via the classical TD-1 method. However, whenever g is satisfied (i.e., $\beta_g(s_{t+1})$ =true), it is updated with the target value given by $V^\phi(\cdot;\xi)$ which can quickly propagate discounted return (reward information) of satisfying ξ back to s_{t+1}, achieving the global optimality of satisfying both g and ξ. Then $Q_g^\theta(\cdot,\cdot;\xi)$ can be updated by minimizing the loss as,

$$J(\theta) = \mathbb{E}_{(s,a,r,s',g,\xi)\sim\mathcal{B}}\left[\ell(Q_g^\theta(s,a;\xi), Q_g^{\text{target}}(s,a,r,s';\xi))\right] \tag{9}$$

where $\mathcal{B}$ is the replay buffer.

4.5 Algorithms

The algorithms for training and testing are in Algorithm 1 and 2 in Appendix A.7, and the algorithm for subgoal extraction is presented in Algorithm 3. The task generation for performance evaluation is introduced in Appendix A.2. More practical implementation techniques are in Appendix A.6.

4.5.1 Model-free Option Planner

In the high level of TL task execution, we use a model-free planner to determine the next option to execute. As shown in Figure 2(b), it uses the standard model-predictive control (MPC) technique [13]: first extracts all the subgoal sequences satisfying the target task ϕ, then finds the optimal subgoal sequence ξ^* according to multi-step value function $V^\phi(s;\xi)$ and finally determines the option of achieving first subgoal $\xi^*[0]$ conditioned on achieving future ones $\xi^*[1:]$, i.e., $o_{\xi^*[0]}^{\xi^*[1:]}$, to execute. Different from previous MPC planners, we do not need to learn a transition model here since V^ϕ is already trained to predict the expected return of achieving various subgoal sequences ξ. So, although previous model-based planners suffer from compounding errors [37, 38], this issue can be avoided by our proposed model-free planner. Whenever the selected option is successfully finished, the task formula needs to be updated by progression with the symbol of achieved subgoal, as indicated by the red arrow in Figure 2(b).

Although function V^ϕ is trained to predict the return of finite subgoal sequences which have maximum length K, V^ϕ is still generalizable to longer sequences, since the representation of subgoal sequence is extracted by GNN which has strong power of generalization. This is validated by the following experiments.

During the task execution, in order to avoid unsafe symbols, whenever task formula is updated the agent finds a set of unsafe symbols $\mathcal{U}_{\text{unsafe}}$ which can falsify the current task φ, i.e., $\mathcal{U}_{\text{unsafe}} = \{q | q \in \mathcal{G}, \text{prog}(q, \varphi) = \text{false}\}$. In the low level of the task execution framework, actions a_t which can lead the agent too close to symbols in $\mathcal{U}_{\text{unsafe}}$ will be ignored, where the closeness is measured by the action value function $Q_g^{\varnothing}$ for $\forall g \in \mathcal{U}_{\text{unsafe}}$. The details of task execution are presented in Algorithm 2 in Appendix.

4.5.2 Theoretical Justification

In this section, we theoretically justify the proposed model-free MPC planner in a deterministic transition environment which contains all the environments where our empirical results are evaluated later. First define the reward function $R(s, g)$ as the maximum expected return obtained for reaching a single subgoal g starting from state s, where the transition function $\mathcal{T}(\cdot|s, g)$ gives the next state when g is reached with s as the starting state. Both $R(s, g)$ and $\mathcal{T}(\cdot|s, g)$ are obtained by following the optimal policy of reaching g only. Then, denote $R(s, g; \xi)$ as the reward function conditioned on reaching subgoal sequence ξ in the future, where $\mathcal{T}(\cdot|s, g; \xi)$ denotes the state transition function conditioned on reaching ξ in the future. Let Q_π denote the Q-function under any (myopic) subgoal policy π, i.e., $Q_\pi(s, g)$. Note that with a slight abuse of notation we also use Q_π to denote the multi-step Q-function, defined as

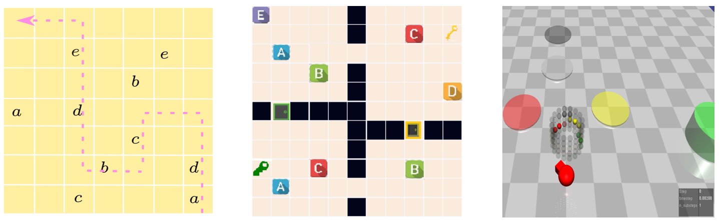
$$\begin{aligned} Q_\pi(s, \tilde{g}_1, \dots, \tilde{g}_K) = \; & \mathbb{E}_{s_{\tau+1} \sim \mathcal{T}(\cdot|s_\tau, g_\tau; g_{\tau+1:K})} \left[\sum_{\tau=1}^{K} R(s_\tau, g_\tau; g_{\tau+1:K}) \middle| g_1 = \tilde{g}_1, \dots, g_K = \tilde{g}_K \right] \\ & + \mathbb{E}_{\substack{g_\tau \sim \pi(\cdot|s_\tau), \\ s_{\tau+1} \sim \mathcal{T}(\cdot|s_\tau, g_\tau)}} \left[\sum_{\tau=K+1}^{\infty} R(s_\tau, g_\tau) \right] \end{aligned} \tag{10}$$

The first term in (10) is equal to the expectation of multi-step value function $V^\phi(s; \tilde{g}_{1:K})$ formulated in Section 4.4, since the MC estimate is unbiased. When the satisfying subgoal sequence extracted from task formula φ is not longer than K, the second term in (10) can be ignored and we have $Q_\pi(s, \tilde{g}_1, \dots, \tilde{g}_K) = V^\phi(s; \tilde{g}_{1:K})$ for any π.

Theorem 1. Let Π be the set of one-step Markov stationary (myopic) policies for subgoal selection, i.e., for any $\pi \in \Pi, \pi : \mathcal{S} \to \Delta(\mathcal{G})$. Suppose the transition dynamics $\mathcal{T}$ of reaching subgoals is deterministic. For any given reward R, denote the K-step subgoal policy obtained from K-step policy improvement as $\pi'_K : \mathcal{S} \to \mathcal{G}^K$, defined as $\pi'_K(s) \in \arg\max_{(g_1,\dots,g_K) \in \mathcal{G}^K} \max_{\pi \in \Pi} Q_\pi(s, g_1, \dots, g_K)$, for all $s \in \mathcal{S}$. Let $V_{\pi'_K}$ denote the value function under the policy π'_K. Then, we have that for all $s \in \mathcal{S}$

$$V_{\pi'_K}(s) \geq \max_{g_{1:K} \in \mathcal{G}^K} \max_{\pi \in \Pi} Q_\pi(s, g_1, \dots, g_K) \geq \max_{g \in \mathcal{G}} \max_{\pi \in \Pi} Q_\pi(s, g) \tag{11}$$

The proof of this theorem is in Appendix A.1. This result shows that the value function of the greedy multi-subgoal policy (π'_K) improves over all the possible K-step subgoal sequences, with the policy after step K to be any policy in Π. Moreover, the value function of π'_K also improves overall one-step policies if the policy after the first step onwards follows any policy in Π. The proposed model-free option planner selects multiple subgoals (at most K subgoals) greedily according to V^ϕ, working same as the greedy multi-subgoal policy π'_K. Note that previous option selection policies for TL task execution [9, 20, 21] are all one-step myopic policies in Π. Therefore, based on Theorem 1, the advantage of the proposed planner over previous ones is theoretically justified.



(a) Letter (b) Room (c) Navigation

Figure 4: Environments. Note that these environments are procedurally generated and hence tasks cannot be solved by simple tabular methods.

5 Experiments

Our experiments are designed to evaluate the performance of multi-task RL agent trained by the proposed framework, including sample efficiency, optimality and generalization. Specifically, we focus on the following questions: 1) **Performance**: whether the proposed framework can outperform previous representative methods in terms of optimality and sample efficiency; 2) **Ablation study**: what is the influence of different components of the proposed framework on the learning performance; 3) **Long horizon tasks**: whether the proposed framework can train the multi-task agent to better solve long-horizon unseen tasks; 4) **Visualization**: what the learned value function looks like for options conditioned on different future subgoals. The neural architecture and hyper-parameters used in experiments are also introduced in the appendix.

5.1 Experiment Setup

We conducted experiments across different environments and TL tasks, where the tasks vary in length and difficulty. All the environments are procedurally generated, where the layout and positions of objects are randomly generated upon reset. The positions and properties of objects are unknown to the agent. As such, none of the environments adopted here can be solved by simple tabular-based methods.

In every training episode, the agent uses appropriate option policies to satisfy a subgoal sequence ξ which is randomly selected from the set of subgoals of the environment. After every fixed number of training steps or episodes, the agent is evaluated on a fixed number of tasks with TL specifications randomly sampled from a large set of possible tasks (more than 10^6). We also evaluate the agent on TL tasks whose solution has longer horizons than subgoal sequences used in the training stage, verifying the generalization of the trained agent to more difficult tasks. The environments are introduced in the following.

Remark. The algorithm performance reported here can be also regarded as the evaluation of zero-shot generalization. First, since the agent is only trained to complete subgoal sequences, the TL tasks in evaluation are unseen in the training. Second, the agent directly applies the policies of trained options to complete tasks in evaluation, so that no further learning is needed for any unseen tasks in the evaluation. These two arguments also hold in baselines. Therefore, the zero-shot generalization capability of the proposed framework is evaluated and compared with baselines.

Letter. This environment is a $n \times n$ grid game which is a variant of that in Figure 1, replacing objects by letters. Out of the n^2 grid cells, m grids are associated with k (where $m > k$) unique propositions (letters). Note that some letters may appear in multiple cells, giving the option of reaching them in multiple ways. An example layout is shown in Figure 4(a) with $n = 7, m = 10$ and $k = 5$. At each step the agent can move along the cardinal directions (up, down, left and right). The agent is given the task specification and is assumed to observe the full grid (and letters) from an egocentric point of view with the image-based observation. Each task is described by a TL formula in terms of these letters. But positions of these letters are unknown to the agent. The agent must visit these letters' locations in certain way to satisfy the TL formula.

Room. This environment is also a grid-world game, but its observation is divided into four rooms as shown in Figure 4(b). There are 5 letters located in 8 positions, corresponding to 5 propositions randomly allocated in these rooms. An example of layout is shown in Figure 4(b). The agent is randomly placed into one of these rooms. Each room is connected to its neighbors by corridors. Two randomly selected corridors are blocked by locks. The agent can open a lock by using a key corresponding to that specific lock (having the same color). These (green and yellow) keys are placed in positions which the agent can reach. This environment is an upgrade of MineCraft with obstacles and dependencies between objects imposed by keys and locks. The observation is also image-based here and the agent does not know the positions of objects. Every task formula is a TL formula in terms of object's letters. The agent must visit these letters' locations in certain way to satisfy the TL formula.

Navigation. This is a robotic environment with continuous action and state spaces. It is modified from OpenAI's Safety Gym [39]. As shown in Figure 4(c), the environment is a 2D plane with 6 to 9 colored circles, called "navigation". Here each color represents a proposition in task specification, with some circles sharing the same color. We use Safety Gym's Point robot whose actions are steering and forward/backward acceleration. Its observation includes the lidar information towards the circles and other sensory data (e.g., accelerometer, velocimeter). The circles and the robot are randomly positioned on the plane at the start of each episode and the robot has to visit and/or avoid certain colors in a particular manner described by the TL specification.

5.1.1 Tasks

We evaluate the proposed framework on two categories of tasks, i.e., DNF and Recursive task. Every category has millions of possible tasks. Every TL task for testing is randomly selected, and the agent does not know any information about the task before learning starts. The details of task generation are introduced in Appendix A.2.

5.1.2 Baselines

The proposed algorithm is compared with three baselines. The model architecture and hyper-parameters of the proposed method and baselines are introduced in Appendix A.5 and A.8. The first baseline (*Option*) is based on the conventional option framework, where every option is only trained to achieve next proposition as the subgoal without considering the future. This baseline is essentially same as methods in [21, 22], whose idea was widely used by previous works on multi-task RL [9, 20–22, 40, 41]. Following [21, 22], in Option baseline the TL task is first decomposed into subgoal sequences and the agent applies pre-trained option policies to achieve subgoals in the sequence one-by-one, where the options were trained myopically without considering future subgoals. In order to make comparisons to be fair, in *Option* baseline the RL algorithms for training the agent and hyper-parameters are the same as the proposed method, where HER and formula transformation are both adopted. However, in *Option* baseline, since options are not conditioned on future subgoals, their training does not need future rewards and the multi-step value function V^{ϕ} is not used.

The second baseline (*GCN-LTL*) is modified from [10], where the task formula is processed by a graph convolutional network (GCN) [42] and progresses over time. The architecture of GCN here is the same as that in [10] with $T = 8$ message passing steps and 32-dimensional node embedding. Other parts of agent's model are the same as the proposed method. The third baseline (*GRU-LTL*) is based on the method in [8]. This approach trains an agent that considers the whole task specification as an extra input and uses GRU [43] to learn an embedding of the TL specification which does not progress over time. The learned task embedding has the size of 32 which is same as the size of embedding of future subgoals in our method. Other parts of agent's model are the same as the proposed method.

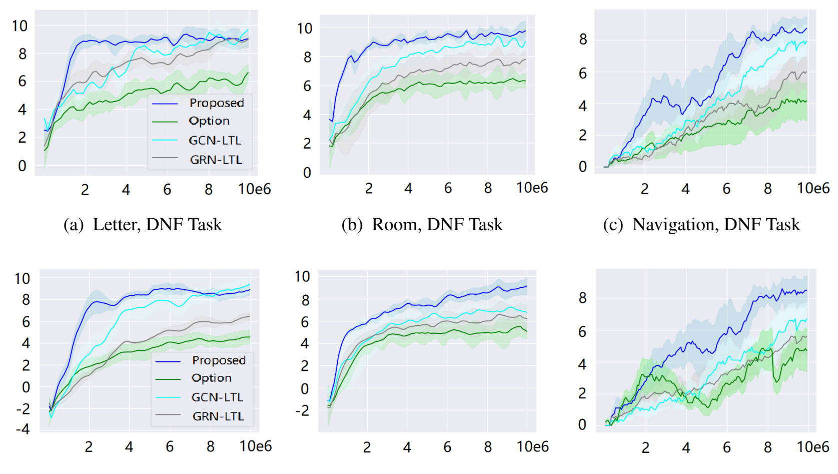
In original papers of GCN-LTL and GRU-LTL [8, 10], the agent is trained by on-policy PPO algorithms. In order to make them comparable with the proposed framework, GCN-LTL and GRU-LTL use same RL algorithm as our framework, with the same hyperparameters as ours. In the letter and room domains, the agents in GCN-LTL and GRU-LTL are trained by the off-policy Q learning [1] approach. In the navigation domain, GCN-LTL and GRU-LTL still use the PPO algorithm. Since the agent takes the original TL specification as its input directly, formula transformation and HER cannot be used in GCN-LTL or GRU-LTL. Since their original implementations are not option-based, the multi-step value function V^{ϕ} is not used either.

5.2 Results

In this section, we present the comparison results of the proposed method with baselines. The overall performance comparisons in terms of average return for satisfying TL tasks are first presented. Then, we demonstrate the ablation studies to investigate the effects of different components of the proposed framework. More ablation study on the multi-step value function in the navigation domain is shown in Section 5.2.2. In each plot of the proposed framework, the x-axis is the number of environment steps used in the option training algorithm, while the y-axis is the evaluation performance of task execution algorithm by applying trained options. In evaluation, the task is randomly generated according to some template.

5.2.1 Performance

In Figure 5, the proposed method is compared with three baselines introduced in Section 5.1.2. We can see that although Baseline-1 can learn fast in the early stage, its overall performance is the worst. The optimality in Baseline-1 degrades because the resulting options myopically focus on the next subgoal only, without looking ahead. It shows the importance of the dependence of options on future subgoals. In addition, the proposed method can learn much faster than GCN-LTL and GRU-LTL, confirming that leveraging reusable skills via options can achieve better sample efficiency. Further, the agents in GCN-LTL and GRU-LTL, which are conditioned



(a) Letter, DNF Task (b) Room, DNF Task (c) Navigation, DNF Task

(d) Letter, Rec. Task (e) Room, Rec. Task (f) Navigation, Rec. Task

Figure 5: Performance Comparisons. The x-axis is the environmental step, and the y-axis is the return. "Rec." is short for recursive. The first row is for evaluating DNF tasks, and the second row is for evaluating recursive tasks. The definitions of DNF and recursive tasks are in Appendix A.2. The return is defined as the sum of rewards along the trajectory.

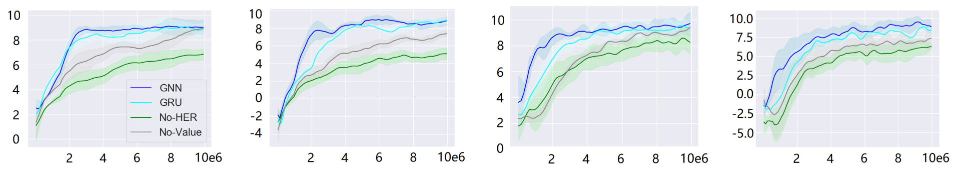
on the task specification directly, need a lot of environment samples to understand temporal operators and find out the optimal path in the formula to finish the task.

5.2.2 Ablation Study

The ablation study is first to examine the difference between GNN and GRU when used in option critics $Q_p^\theta(\cdot, \cdot; \xi)$ and value function $V^\phi(\cdot; \xi)$ to learn the embedding of the sequence ξ of future subgoals. Specifically, the nodes of GNN represent subgoals and every subgoal is connected to its successor by a directed edge. The embedding of sequence ξ is learned by GCN with multi-step message passing ($T = 8$). In addition, when the GRU is used, sequence ξ, with every element one-hot encoded, is fed into GRU and the embedding can be obtained at the output of GRU. More details of agent's model are in Appendix. In Figure 6, we can see that the GRU performs slightly worse than GNN.

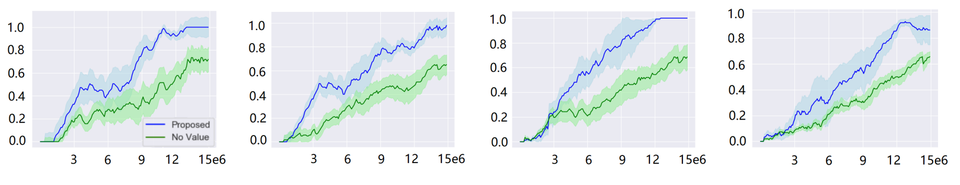
In addition, we study the effects of multi-step value function V^ϕ and HER by comparing "No-value" and "No-HER" with the proposed method in Figure 6. We can see that when V^ϕ or HER is not used, the learning performance can degrade significantly, implying their importance in performance improvement. Since the time horizon of tasks in the navigation domain is much longer than other domains, we find that the multi-step value function V^ϕ plays a more important role in navigation domain. The reward propagation is more difficult when task's time horizon increases, and hence the usage of V^ϕ can improve the sample efficiency of our framework significantly in this case.

Multi-step Value Function. In navigation domain, the time horizon of every task is 1000 which is much longer than that in other domains. The experiments in this section are conducted in two environments of navigation domain. The first environment has 3 colors and 6 objects, denoted as "Nav1" whereas the second environment has 5 colors and 10 objects, denoted as



(a) Letter, DNF Task (b) Letter, Rec. Task (c) Room, DNF Task (d) Room, Rec. Task

Figure 6: Ablation study. The x-axis is the environmental step, and the y-axis is the return. "No-HER" refers to the proposed method without using HER. "No-value" refers to the proposed method without using the multi-step value function.



(a) Nav1, DNF Task (b) Nav2, DNF Task (c) Nav1, Rec. Task (d) Nav2, Rec. Task

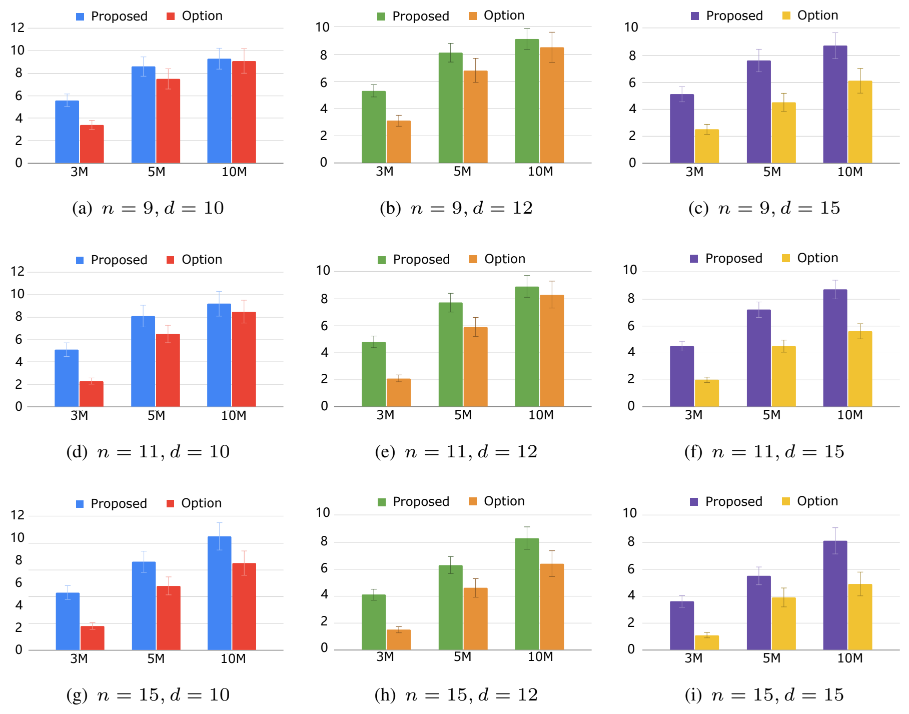
Figure 7: The ablation study of the multi-step value function V^{ϕ} in navigation domain. The x-axis is the environmental step, and the y-axis is the success rate.

"Nav2". Both DNF and Recursive tasks are evaluated in these two environments. The results of ablation study on the multi-step value function V^{ϕ} are shown in Figure 7, where the curve of "No Value" refers to our framework without using V^{ϕ}. Compared with other experiment results, we can see that V^{ϕ} can improve the sample efficiency more in navigation domain.

5.3 Long Horizon Tasks

In order to verify the effectiveness of the reward propagation, we evaluate the performance of the trained RL agent in tasks with long time horizon. We focus on the letter domain where the map size and the depths of the TL specification are changed for comparison. The depth of a formula φ is the length of optimal subgoal sequence to satisfy φ. Baseline Option only learns independent option for each subgoal and does not consider reward propagation. Baseline GRU-LTL uses recurrent GNN to process the TL specification not progressed, so its performance on TL tasks with long horizon is much worse than Baseline GCN-LTL. Therefore, we do not consider Baselines GCN-LTL and GRU-LTL for comparison here. In every experiment, there are 8 unique letters on the map and every letter appears twice.

The comparison results in terms of episodic return are shown in Figure 8. Since the evaluation results of long-horizon tasks have large variances, we only show the results as charts here. The TL specification for evaluation is a DNF task consisting of 3 conjunctions with the depth of d, where every letter is randomly generated without repetition. Every task here has longer horizon than that in Figure 5. Every result in Figure 8 is the average of 10 formulas, and the variance is obtained from 5 seeds. We can see that the proposed method can significantly outperform the Baseline Option. Moreover, the superiority of our proposed



(a) $n = 9, d = 10$ (b) $n = 9, d = 12$ (c) $n = 9, d = 15$

(d) $n = 11, d = 10$ (e) $n = 11, d = 12$ (f) $n = 11, d = 15$

(g) $n = 15, d = 10$ (h) $n = 15, d = 12$ (i) $n = 15, d = 15$

Figure 8: Performance comparison for long-horizon tasks in letter domain. The x-axis is the number of environment steps taken for option training. The y-axis is the average sum of rewards received in the trajectory. The map size is $n \times n$ and the task formula has depth of d. The evaluation takes place at the steps of $\{3, 5, 10\} \times 10^6$, during the option training.

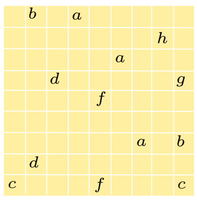
method improves with map size and formula depth, showing that the proposed method can solve the long-horizon tasks well and the effect of reward propagation is significant.

5.4 Visualization

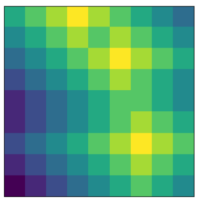
Finally, in order to show the effects of the dependence of options on future subgoals, we visualize the Q functions of the same option dependent on different future subgoals. The color of very grid represents the discounted return to the target subgoal, where the brighter the color is, the higher the return will be.

In Figure 9, the first row shows the Q functions of reaching subgoal a in letter domain in three scenarios, namely dependent on nothing, b and $b \rightarrow c$. Every grid represents the environment state where the agent is in that grid. On the map shown in Figure 9(a), there are three letters a. According to Figure 9(b), the agent should go to the closest a. Figures 9(c) and 9(d) tell us that when dependent on b or $b \rightarrow c$, the option of reaching a regards a in first row or 7-th row as the target.

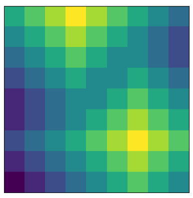
In the second row of Figure 9, we can see that in room domain, the option of reaching C has different targets when the future subgoal sequences ξ are different. Specifically, In Figure 9(h), the grid containing the yellow key has the highest value in the bottom rooms and the grid



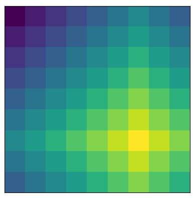
(a) Letter



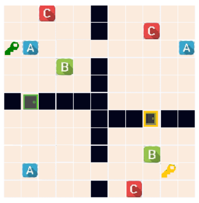
(b) $Q_a^\theta(\cdot, *; \varnothing)$



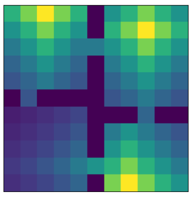
(c) $Q_a^\theta(\cdot, *; b)$



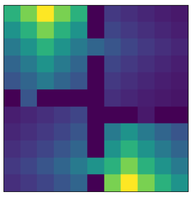
(d) $Q_a^\theta(\cdot, *; b, c)$



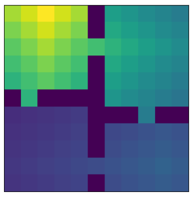
(e) Room



(f) $Q_C^\theta(\cdot, *; \varnothing)$



(g) $Q_C^\theta(\cdot, *; B)$



(h) $Q_C^\theta(\cdot, *; B, A)$

Figure 9: Visualization of trained action-value function of options. The first row is for the option of reaching a in letter domain, and the second row is for the option of reaching C in room domain. The color in every grid (state s) corresponds to the Q value of the optimal action, i.e., $\forall s, Q_p^\theta(s, *; \xi) = max_a Q_p^\theta(s, a; \xi)$.

having C in the upper left room has the highest value across the whole map. This indicates that in environment states where the agent is in the bottom rooms, the agent should first go to pick up the yellow key as an intermediate target and then go to C in the upper left corner. It shows that the agent successfully learns the skill of opening a lock by the right key, without having any key proposition or prior knowledge.

6 Conclusion

In this work, we propose a novel framework for generalizing TL tasks by options dependent on the future subgoal sequence. Moreover, to facilitate the reward propagation of satisfying future subgoals, we propose to learn a multi-step value function updated by Monte Carlo estimates of discounted return. Based on these, we also propose a new model-free option planner for task execution, which circumvents the compounding errors caused by the learned transition model. With comprehensive experiments, the proposed method is confirmed to have significant advantages over previous methods in terms of optimality and sample efficiency.

The limitation of this proposed framework is that it cannot solve TL tasks with noisy symbolic observations [44]. In real-world applications the values of symbols or propositions can be stochastic, and the achievement of subgoals and even tasks can be probabilistic. In the future work, we will extend the proposed framework to handle TL task with noisy symbolic observations by learning a probabilistic model of symbols conditioned on the input observation.

7 Declarations

7.1 Funding

This study was funded by National Science Foundation (NSF) under Award No. 1837369.

7.2 Conflicts of Interests

Not applicable.

7.3 Ethics Approval

Not applicable.

7.4 Consent to Participate

Not applicable.

7.5 Consent for Publication

Not applicable.

7.6 Availability of Data and Material

Not applicable.

7.7 Code Availability

Not applicable.

7.8 Authors' Contributions

D.X. formulate the problem, designs the algorithm, conducts the experiments and writes the paper. F.F. revises the paper and provides the supervision.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., *et al.*: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)

[2] Badia, A.P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z.D., Blundell, C.: Agent57: Outperforming the atari human benchmark. In: International Conference on Machine Learning, pp. 507–517 (2020). PMLR

[3] Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research **17**(1), 1334–1373 (2016)

[4] Inala, J.P., Ma, Y.J., Bastani, O., Zhang, X., Solar-Lezama, A.: Safe human-interactive control via shielding. arXiv preprint arXiv:2110.05440 (2021)

[5] De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI'13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 854–860 (2013). Association for Computing Machinery

[6] Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Teaching multiple tasks to an rl agent using ltl. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 452–461 (2018)

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

[8] Kuo, Y.-L., Katz, B., 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). IEEE

[9] Araki, B., Li, X., Vodrahalli, K., DeCastro, J., Fry, M., Rus, D.: The logical options framework. In: International Conference on Machine Learning, pp. 307–317 (2021). PMLR

[10] Vaezipoor, P., Li, A.C., Icarte, R.A.T., Mcilraith, S.A.: Ltl2action: Generalizing ltl instructions for multi-task rl. In: International Conference on Machine Learning, pp. 10497–10508 (2021). PMLR

[11] Hengst, F., François-Lavet, V., Hoogendoorn, M., Harmelen, F.: Reinforcement learning with option machines. In: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, pp. 2909–2915 (2022). International Joint Conferences on Artificial Intelligence Organization

[12] Liu, J.X., Shah, A., Rosen, E., Konidaris, G., Tellex, S.: Skill transfer for temporally-extended task specifications. arXiv preprint arXiv:2206.05096 (2022)

[13] Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: Theory and practice—a survey. Automatica **25**(3), 335–348 (1989)

[14] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, ??? (2018)

[15] Camacho, A., Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI (2019)

[16] Icarte, R.T., Klassen, T., Valenzano, R., McIlraith, S.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: International Conference on Machine Learning, pp. 2107–2116 (2018). PMLR

[17] Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. Journal of Artificial Intelligence

Research **73**, 173–208 (2022)

[18] Littman, M.L., Topcu, U., Fu, J., Isbell, C., Wen, M., MacGlashan, J.: Environment-independent task specifications via gltl. arXiv preprint arXiv:1704.04341 (2017)

[19] Jothimurugan, K., Bansal, S., Bastani, O., Alur, R.: Compositional reinforcement learning from logical specifications. Advances in Neural Information Processing Systems **34**, 10026–10039 (2021)

[20] Andreas, J., Klein, D., Levine, S.: Modular multitask reinforcement learning with policy sketches. In: International Conference on Machine Learning, pp. 166–175 (2017). PMLR

[21] León, B.G., Shanahan, M., Belardinelli, F.: Systematic generalisation through task temporal logic and deep reinforcement learning. arXiv preprint arXiv:2006.08767 (2020)

[22] León, B.G., Shanahan, M., Belardinelli, F.: In a nutshell, the human asked for this: Latent goals for following temporal specifications. In: International Conference on Learning Representations (2021)

[23] Chrisman, L.: Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In: AAAI, vol. 1992, pp. 183–188 (1992). Citeseer

[24] Shani, G., Brafman, R.: Resolving perceptual aliasing in the presence of noisy sensors. Advances in Neural Information Processing Systems **17** (2004)

[25] Shani, G.: A survey of model-based and model-free methods for resolving perceptual aliasing. Ben-Gurion University (2004)

[26] Lajoie, P.-Y., Hu, S., Beltrame, G., Carlone, L.: Modeling perceptual aliasing in slam via discrete–continuous graphical models. IEEE Robotics and Automation Letters **4**(2), 1232–1239 (2019)

[27] Liu, M., Zhu, M., Zhang, W.: Goal-conditioned reinforcement learning: Problems and solutions. arXiv preprint arXiv:2201.08299 (2022)

[28] Li, S., Zhang, J., Wang, J., Yu, Y., Zhang, C.: Active hierarchical exploration with stable subgoal representation learning. arXiv preprint arXiv:2105.14750 (2021)

[29] Chane-Sane, E., Schmid, C., Laptev, I.: Goal-conditioned reinforcement learning with imagined subgoals. In: International Conference on Machine Learning, pp. 1430–1440 (2021). PMLR

[30] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al.: Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905 (2018)

[31] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30 (2016)

[32] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[33] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2), 181–211 (1999)

[34] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp. 46–57 (1977). ieee

[35] Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning About Systems. Cambridge university press, ??? (2004)

[36] Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial intelligence **116**(1-2), 123–191 (2000)

[37] Asadi, K., Misra, D., Littman, M.: Lipschitz continuity in model-based reinforcement learning. In: International Conference on Machine Learning, pp. 264–273 (2018). PMLR

[38] Lambert, N., Pister, K., Calandra, R.: Investigating compounding prediction errors in learned dynamics models. arXiv preprint arXiv:2203.09637 (2022)

[39] Ray, A., Achiam, J., Amodei, D.: Benchmarking safe exploration in deep reinforcement learning. arXiv preprint arXiv:1910.01708 **7**, 1 (2019)

[40] Sohn, S., Oh, J., Lee, H.: Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. Advances in Neural Information Processing Systems **31** (2018)

[41] Sun, S.-H., Wu, T.-L., Lim, J.J.: Program guided agent. In: International Conference on Learning Representations (2019)

[42] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)

[43] Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)

[44] Li, A.C., Chen, Z., Vaezipoor, P., Klassen, T.Q., Icarte, R.T., McIlraith, S.A.: Noisy symbolic abstractions for deep rl: A case study with reward machines. In: Deep Reinforcement Learning Workshop NeurIPS 2022 (2022)

[45] Schlichtkrull, M., Kipf, T.N., Bloem, P., Berg, R.v.d., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European Semantic Web Conference, pp. 593–607 (2018). Springer

[46] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., Zaremba, W.: Hindsight experience replay. Advances in

neural information processing systems **30** (2017)

[47] Xu, D., Fekri, F.: Generalizing ltl instructions via future dependent options. arXiv preprint arXiv:2212.04576 (2022)

Appendix A Appendix

A.1 Proof of Theorem 1

In this section, we present the proof of Theorem 1 in Section 4.5.2. Define

$$Q_K^{\max}(s, g_1, \dots, g_K) := \max_{\pi \in \Pi} Q_\pi(s, g_1, \dots, g_K), \text{ and } Q^{\max}(s, g) := \max_{\pi \in \Pi} Q_\pi(s, g) \tag{A1}$$

We define the Bellman operator under the multi-step policy π'_K (open-loop) as follows: for any $Q \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{G}|^K}$

$$\mathcal{T}_{\pi'_K}(Q)(s, g_1, \dots, g_K) = \mathbb{E}\Big[\sum_{k \in [K]} \gamma^{k-1} R(s_k, g_k; g_{k+1:K}) + \gamma^K Q(s_{K+1}, \pi'_K(s_{K+1})) \Big| s_1 = s, g_{1:K}\Big] \tag{A2}$$

Note that $\mathcal{T}_{\pi'_K}$ is a contracting operator, and we denote the fixed point of this operator as $Q_{\pi'_K} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{G}|^K}$, which is the Q-function of the multi-step policy π'_K. According to the definition [14], we also know the its state-value function $V_{\pi'_K}(s) = Q_{\pi'_K}(s, \pi'_K(s))$ which applies the multi-step policy π'_K to subgoals $g_{1:K}$ in $Q_{\pi'_K}(s, g_{1:K})$. Then, we have

$$\mathcal{T}_{\pi'_K}(Q_K^{\max})(s, g_1, \dots, g_K) = \mathbb{E}\Big[\sum_{k \in [K]} \gamma^{k-1} R(s_k, g_k; g_{k+1:K}) + \gamma^K Q_K^{\max}(s_{K+1}, \pi'_K(s_{K+1})) \Big| s_1 = s, g_{1:K}\Big]$$

$$= \mathbb{E}\Big[\sum_{k \in [K]} \gamma^{k-1} R(s_k, g_k; g_{k+1:K}) + \gamma^K \max_{g_{K+1:2K}} Q_K^{\max}(s_{K+1}, g_{K+1:2K}) \Big| s_1 = s, g_{1:K}\Big] \tag{A3}$$

$$\geq \mathbb{E}\Big[\sum_{k \in [K]} \gamma^{k-1} R(s_k, g_k; g_{k+1:K}) + \gamma^K \max_{g_{K+1:2K}} Q_\pi(s_{K+1}, g_{K+1:2K}) \Big| s_1 = s, g_{1:K}\Big] \tag{A4}$$

$$\geq \mathbb{E}\Big[\sum_{k \in [K]} \gamma^{k-1} R(s_k, g_k; g_{k+1:K}) + \gamma^K Q_\pi(s_{K+1}, \pi(s_{K+1}), \dots, \pi(s_{2K})) \Big| s_1 = s, g_{1:K}\Big] \tag{A5}$$

$$= Q_\pi(s, g_1, \dots, g_K) \tag{A6}$$

for any $\pi \in \Pi$, where (A3) is due to the definition of π'_K, (A4) is from the definition of $Q_K^{\max}$, (A5) is from the max operator, and (A6) uses the definition. Therefore, because of the monotonicity of $\mathcal{T}_{\pi'_K}$ shown above, we have

$$Q_{\pi'_K}(s, g_{1:K}) = \lim_{n \to \infty} (\mathcal{T}_{\pi'_K})^n (Q_K^{\max})(s, g_{1:K}) \geq Q_K^{\max}(s, g_{1:K}) \geq \max_{\pi \in \Pi} Q_\pi(s, g_{1:K}) \tag{A7}$$

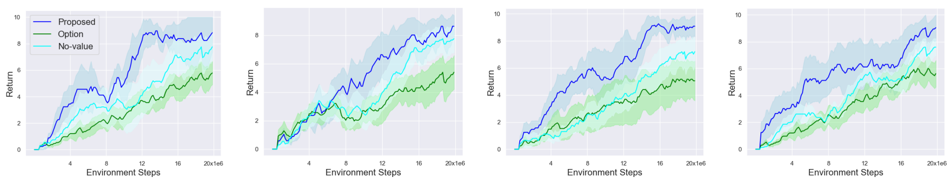
Applying π'_K on both sides of (A7) yields,

$$V_{\pi'_K}(s) = Q_{\pi'_K}(s, \pi'_K(s)) \geq \max_{\pi \in \Pi} Q_\pi(s, \pi'_K(s)) = \max_{g_{1:K}} \max_{\pi \in \Pi} Q_\pi(s, g_{1:K}) \tag{A8}$$

Since the multi-step maximization is not smaller than single-step maximization, we have,

$$\max_{g_{1:K}} \max_{\pi \in \Pi} Q_\pi(s, g_{1:K}) \geq \max_{g} \max_{\pi \in \Pi} Q_\pi(s, g) \tag{A9}$$

which can prove the theorem in the combination with (A8).



(a) Nav1, DNF Task (b) Nav1, Rec Task (c) Nav2, DNF Task (d) Nav2, Rec Task

Figure A1: Performance evaluation in Navigation environment with Car agent. Nav1 and 2 are different setups of navigation environment, introduced in Section 5.2.2.

A.2 Task Generation

We define the *depth* of a task formula φ as the length of the shortest subgoal sequence to satisfy φ. We generate two kinds of tasks to evaluate agent's performance.

The first kind of task is the "DNF" task described by a disjunctive normal formula that concatenates terms by disjunctive operator $\cup$, i.e., $\varphi_{dnf} = \varphi_{dnf} \cup \varphi'$ and $\varphi' = \varphi'; s|\varphi'; \neg g$. Here, s and g are propositions denoting two different subgoals. The notation $|$ denotes alternative. When generating a task formula, two sub-formulae around $|$ are randomly selected. For instance, $\varphi_{\text{dnf}} = (a; b; \neg e) \cup (c; d)$. Specifically, the number of terms that are connected with the disjunctive operator ranges between 3 and 5, and the number of propositions in every term is between 1 to 5.

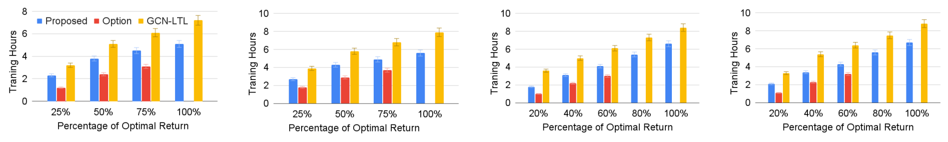
The second type of task is called "Recursive" task, which can be formulated as $\varphi_{\text{rec}} = \varphi_{\text{rec}}; \varphi' | \varphi_{\text{rec}} \cap \varphi'$ and $\varphi' = g \vee \varphi' | \neg s \mathcal{U} (g \vee \varphi') | \neg s \mathcal{U} g$. Here, s and g are propositions denoting two different subgoals. The depth of the recursion is randomly selected between 3 and 5. An example of "recursive" task is $(\neg a \cup (b \vee c)); e; (\neg f \cup g)$, and the shortest subgoal sequence for satisfying this task is $b \rightarrow e \rightarrow g$ or $c \rightarrow e \rightarrow g$ with the depth of 3.

A.3 Additional Results in Navigation Environment

In addition to Section 5.2.1, we conduct more experiments in Navigation environment with more complex agent of car. This agent simulates a wheeled robot with differential drive control. The performance evaluation is shown in Figure A1. The setups of Nav 1 and 2 are same as that introduced in Section 5.2.2. Here, we compare the proposed framework with *Option* baseline which learns options of reaching different subgoals myopically without looking into the future, similar as that in [21, 22]. Then the proposed framework without multi-step value function is also evaluated, denoted as *No-value*. We can see that, in all these experiments, the option baseline performs the worst, showing the importance of future subgoals. The no-value baseline can reach similar performance as the proposed framework, but it has a much slower convergence speed, demonstrating the effect of multi-step value function on accelerating the value propagation and learning speed.

A.4 Comparison of Time and Memory Complexity

In this section, we evaluate the proposed framework for the CPU time and memory consumption in letter and navigation domains. The computer where the experiments are conducted has Intel i5 CPU, 32G memory, and a 3090 Ti GPU. The tasks for evaluation have long horizon,



(a) Letter, DNF Task (b) Letter, Rec Task (c) Nav, DNF Task (d) Nav, Rec Task

Figure A2: Time comparison of the proposed framework and baseline methods. The experiments are conducted in letter and navigation (Nav) domains with DNF and Rec tasks evaluated. We compare the amount of training hours used by different methods to achieve the same performance in terms of episodic return. The missing values for Baseline Option are due to the fact that the corresponding performance cannot be achieved by this method.

with the depth of $d = 10$. The baselines compared here are Option and GCN-LTL. For time comparison, although the CPU time for every environmental step is similar, different algorithms still need different amount of time to achieve the same performance. So, we compare the proposed framework with baselines on the amount of consumed time for achieving the same evaluation performance. The metric of performance here is the average episodic return of evaluated tasks.

In Figure A2, defining "optimal return" as the best performance achieved by evaluated methods, we compare the amount of training hours used by proposed and baseline methods to achieve the same percentage of optimal return. We can see that the Option has the best time efficiency, since it only trains options independently without considering the future. But it cannot achieve the same optimal performance as other methods in terms of episodic return and hence sacrifices the optimality. The GCN-LTL method can achieve same optimal return as the proposed one, but it has poor time efficiency, since it does not train or use options as reusable skills. In navigation domain, the gap of time consumption is larger between the proposed and GCN-LTL, since the control of simulated robot is more difficult in this environment.

Table A1: Memory Complexity Comparison (unit: MB)

	Letter	Nav.
Proposed	2403	2237
Option	2135	1969
GCN-LTL	2358	2172

In Table A1, we also compare the memory complexity of different methods. We only focus on GPU memory since every experiment is conducted on GPU. Memory is primarily consumed by the agent's model and it is not related with evaluation tasks. The memory consumption in Letter domain is larger than that in Navigation domain. This is because the observation in Letter is processed by a CNN as images, and the observation in Navigation is status of the robot and processed by an MLP. The Option method has smallest memory consumption, since it trains every option myopically with simple value function. The GCN-LTL has similar memory consumption as the proposed one.

A.5 Neural Network Architecture

The agent's architecture of critic (Q or V function) is shown in Figure A3. The input consists of observation, subgoal embedding and subgoal sequence. The observation is processed by the perception module. The subgoal embedding is the one-hot encoding of the subgoals in $\mathcal{G}$. The future subgoal sequence is processed by GNN or GRU. After inputs are processed, the embeddings of observation, subgoal and future subgoal sequence are concatenated and fed into an MLP to predict the return. The multi-step value function V^{ϕ} has the same architecture as the critic function, except that its inputs are only the observation and future subgoal sequence.

The perception module is determined by the observation space of the environment. In letter/room domain with map size of $n \times n$, we used a 3-layer convolutional neural network (CNN) which have 16, 32 and 64 channels, respectively, where the kernel size is $l \in \{2, 3, 4\}$ and stride is 1. In navigation domain, we used a 2-layer fully-connected network with [256, 256] units and ReLU activations.

The sequence of future subgoal is processed by GNN or GRU here. The GNN used here is a graph convolutional network (GCN) [42, 45] with 8 message passing steps and 32-dimensional node embeddings. The GRU used here is a 2-layer bidirectional GRU with a 32-dimensional hidden layer.

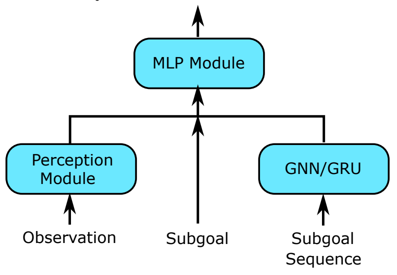


Figure A3: Neural Architecture of $Q_p^{\theta}(\cdot, \cdot; \xi)$ or $V_p^{\theta}(\cdot; \xi)$, where p is the subgoal and ξ is the future subgoal sequence.

For the MLP part of the critic function in Figure A3, we use 3 fully-connected layers with [64, 64, d_a] units and ReLU activations for all three domains. For discrete action space environments, a_d is the number of possible actions, and the output of critic function was passed through a logit layer before softmax. For the continuous case, a_d is the action dimension and we also need to train an actor network sharing same architecture as the critic network except the Tanh activation. Then we assume a Gaussian action distribution and parameterized its mean and standard deviation by sending the actor's output to two separate linear layers.

In three baselines, the Q/value networks and actor network of the agent have the same architectures introduced here, keeping the same model complexity as the proposed method. In baseline Option, since the option does not consider future subgoals, the critic network does not have any module to process the subgoal sequence. In baselines GCN-LTL and GRU-LTL, since they do not use options, the critic network and actor network do not have any subgoal as its input, where TL specification is first transformed into a syntax tree and processed by a GCN (in baseline GCN-LTL) or GRU (in baseline GRU-LTL without progression). The GCN has the same architecture as that introduced above. The GRU in baseline-3 is also a 2-layer bidirectional GRU with 32-dimensional hidden layers.

A.6 Practical Implementation Techniques

Training Curriculum. In the option training, the agent is trained to satisfy a randomly generated subgoal sequence ξ with maximal environment return. Denote the maximum length of ξ as K. The training curriculum consists of K levels. As such, in the k-th level ($k = 1, \dots, K$), the length of subgoal sequence ξ is set to be k. Whenever the average success rate in k-th level is above a threshold (e.g., 80%), the agent will proceed to $(k + 1)$-th level where the length of subgoal sequences becomes longer. Therefore, the difficulty of tasks increases gradually as the agent proceeds to higher levels. For any subgoal sequence ξ, the agent applies options to satisfy subgoals in ξ one-by-one with conditions of future subgoals. The details are introduced in Algorithm 1. In letter and room domains, we use deep Q learning [1] (off-policy) to train options, whereas in the navigation domain, we use PPO [32] algorithm (on-policy) to train options. The details of hyper-parameters are introduced in Section A.8.

Adversarial Scheme. We also adopt an adversarial scheme for selecting training options which can improve the learning efficiency in empirical experiments. In the k-th level, at the beginning of each episode with initial state s_0, multiple subgoal sequences with the same length are randomly generated, i.e., $\{\xi_i\}_{i=1}^{N_T}$, and the j-th sequence with the lowest value is selected as the training task for the agent, i.e., $j = \arg\min_{i=1,\ldots,N_T} V^{\phi}(s_0;\xi_i)$. This implies that a difficult task in the current level is selected to train the agent, always pushing forward the capability of the learning agent.

Hindsight Experience Replay In early learning stage, most trajectories produced by agent's policies cannot achieve or satisfy the given task, which cannot provide any useful reward information to train agent's policy and value functions. Therefore, in training the options, in order to improve the learning efficiency, we propose to modify the hindsight experience replay (HER) [46] to better utilize the past unsuccessful trajectories. We extend HER to temporal logic domain by modifying any unsuccessful trajectory whose given task was not successfully finished. Specifically, in any unsuccessful trajectory τ associated with subgoal sequence ξ (ξ is not finished by τ), we find ξ' which is the subgoal sequence satisfied by τ actually and replace ξ by ξ', so that the trajectory τ associated with ξ' becomes a successful trajectory (ξ' is satisfied by τ). Then, assigning a large positive reward R_F at the time step when $\xi'[-1]$ becomes satisfied, designating the trajectory τ successful and hence useful to the training.

Table A2: Hyperparameters of PPO in Navigation Domain

Hyperparameter	Value
Env. steps per update	2048
Minibatch size	256
Discount	0.995
Time horizon of an episode	1000
Total number of steps	15e6
Satisfaction Reward R_F	10
HER trajectory modification ratio	1.0
Evaluation interval (episodes)	100
Evaluation episodes	10
Optimizer	Adam
Learning rate	3×10^{-4}
GAE-λ	0.95
Entropy coefficient	0.01
Value loss coefficient	0.5
Gradient clipping	0.5
PPO clipping (ϵ)	0.2

A.7 Algorithms

We summarize the detailed operations in option training and task execution in Algorithms 1 and 2, respectively. Algorithm 1 trains the Q function in line 21 via off-policy method, which can be trivially extended to train V function via on-policy method. The algorithm of extracting subgoal sequences from the TL task is presented in Algorithm 3.

A.8 Hyper-parameters

In the proposed framework, we use deep Q learning [1] to learn options in letter and room domains, while we use PPO [32] in the navigation domain. All experiments were conducted on a compute cluster using 1 GPU (RTX 2080 Ti). The hyper-parameters used for deep Q learning in letter and room domain are introduced in Table A3. The hyper-parameters for PPO

Algorithm 1 Option Training Algorithm

```
1: Environment MDP $\mathcal{M}_e$; labeling function $L$; positive reward for task completion $R_F$; The set of propositions $\mathcal{P}$ and subgoals $\mathcal{G}$; multi-step value function $V^{\phi}(s;\xi)$; Q function of option policy $Q_g^{\theta}(s,a;\xi)$ for $\forall g \in \mathcal{G}$; the subgoal planner $\Gamma^{\vartheta}$; replay buffer $\mathcal{B}$; trajectory buffer $\mathcal{B}_t$; episodic buffer $\mathcal{E}$; maximum length of subgoal sequence $K$; performance threshold $\zeta$ of upgrading to next level
2: Initialize parameters $\theta, \vartheta$ and $\phi$;
3: Initialize $\mathcal{B} \leftarrow []$;
4: % levels from 1 to $K$;
5: for $k = 1, \ldots, K$ do
6:     % train the options;
7:     while the average success rate is below $\zeta$ do
8:         Initialize $\mathcal{E} \leftarrow []$;
9:         Reset environment $s \leftarrow s_0$;
10:        Randomly generate $N_S$ subgoal sequences, and select $\xi$ with lowest value on $V^{\phi}$;
11:        for $l = 1, \ldots, \text{len}(\xi)$ do # len($\xi$) denotes the length of $\xi$
12:            $\tilde{s}_0 \leftarrow s$;
13:            for $t = 0, \ldots, T_S - 1$ do
14:                Apply option policy $\pi_{\xi[0]}^{\xi[1:]}$ into the environment $\mathcal{M}_e$;
15:                Obtain reward $r_t$ and next state $\tilde{s}_{t+1}$;
16:                Store experience tuple $(\tilde{s}_t, a_t, r_t, \tilde{s}_{t+1}, \xi[0], \xi[1:])$ into $\mathcal{E}$ and $\mathcal{B}$;
17:                if $L(\tilde{s}_{t+1}) \models \xi[0]$ then
18:                    Set $s \leftarrow \tilde{s}_{t+1}$ and $\xi \leftarrow \xi[1:]$;
19:                    Go to 10;
20:                end if
21:                Sample a minibatch $\mathcal{B}_M$ from $\mathcal{B}$ and update Q function according to (9);
22:                Sample trajectories from $\mathcal{B}_t$ and update $V^{\phi}$ according to (7);
23:            end for
24:            Break; # the trajectory $\mathcal{E}$ is unsuccessful and needs to be relabeled
25:        end for
26:        if $\mathcal{E}$ is unsuccessful then # relabel unsuccessful trajectory by HER
27:            Randomly select subgoal sequence $\xi'$ satisfied by $\mathcal{E}$;
28:            Relabel the subgoal and condition (future subgoal) of every tuple in $\mathcal{E}$ based on $\xi'$;
29:        end if
30:        Store transitions of $\mathcal{E}$ into $\mathcal{B}$;
31:        Store $\mathcal{E}$ into $\mathcal{B}_t$;
32:    end while
33: end for
```

in navigation domain are presented in Table A2. The agents in three baselines are trained by the same RL algorithms in the proposed method, using the same algorithm hyperparameters of the proposed method. In baseline Option, we do not consider any future subgoal sequence. In baselines GCN-LTL and GRU-LTL, we cannot use TTL transformation or HER since the TL specification is transformed into a syntax tree from its original form.

A.9 Discussion

We have a preprint version of this paper uploaded on arXiv [47]. There are significant differences between [47] and this work. First, in order to improve computing efficiency in evaluation,

Algorithm 2 Task Execution Algorithm

```
1: The environment $\mathcal{M}_e$; labeling function $L$; the set of propositions $\mathcal{P}$; progression function prog$(\cdot,\cdot)$ introduced in [22]; multi-step value function $V^{\phi}$ and critics of options $Q_g^{\theta}$ for $\forall g \in \mathcal{G}$ trained by Algorithm 1; the threshold of closeness $\kappa$; the test task specification $\varphi$;
2: Reset environment and obtain the initial state $s_0$;
3: Transform task specification $\varphi$ into a set $\mathcal{K} = \{\xi_i\}_{i=1}^{M_\varphi}$ of accepting subgoal sequences by using Algorithm 3;
4: Given $\varphi$, obtain the set of unsafe subgoals $\mathcal{U}_s$;
5: Select $\xi^*$ with largest value such that $\xi^* = \arg\max_{\xi \in \mathcal{K}} V^{\phi}(s_0; \xi)$;
6: set $t \leftarrow 0$;
7: while every sequence $\xi \in \mathcal{K}$ is not empty do
8:     Sample action $a_t$ from the option policy $\pi_{\xi^*[0]}^{\xi^*[1:]}(\cdot|s_t)$ until $\forall g \in \mathcal{U}_s, Q_g^{\theta}(s_t, a_t; \varnothing) < \kappa$
9:     Obtain next state $s_{t+1}$;
10:    if $L(s_{t+1}) \models \xi^*[0]$ then
11:        Progress the formula $\varphi \leftarrow$ prog$(L(s_{t+1}), \varphi)$
12:        Update the set $\mathcal{U}_s \leftarrow \{q | q \in \mathcal{P}_G$, prog$(q, \varphi) =$ false$\}$;
13:        $\forall \xi \in \mathcal{K}$, if $L(s_{t+1}) \models \xi[0]$, then $\xi$.pop$(\xi[0])$
14:        Select again $\xi^* = \arg\max_{\xi \in \mathcal{K}} V^{\phi}(s_{t+1}; \xi)$;
15:    end if
16:    $t \leftarrow t + 1$
17: end while
```

Algorithm 3 Transforming task specification into a list of subgoal sequences [22]

```
1: Task specification $\varphi$; the set of propositions $\mathcal{P}$;
2: Initialize $\mathcal{K} \leftarrow \{\}$;
3: for each atomic task $p \in \varphi$ do
4:     if $p$ is atomic positive or negative then
5:         for all Seq $\in \mathcal{K}$ do
6:             Seq.append($p$)
7:         end for
8:     else
9:         # There are non-determinstic choices
10:        Initialize choice list: CL
11:        LK $\longleftarrow$ len($\mathcal{K}$)
12:        for all Seq $\in \mathcal{K}$ do
13:            for all choice $\in p$ do
14:                CL.append(choice)
15:                Generate a clone per choice Seq$'$ $\longleftarrow$ Seq
16:                $\mathcal{K}$.append(Seq')
17:            end for
18:        end for
19:        Initialize counter $c \longleftarrow -1$
20:        for $i = 1, 2, \ldots,$ len($\mathcal{K}$) do
21:            if $i$%LK $==$ 0 then
22:                $c+ = 1$
23:            end if
24:            We append a different choice to each sequence cloned $\mathcal{K}[i]$.append(CL$[c]$)
25:        end for
26:    end if
27: end for
28: Return $\mathcal{K}$
```

this work proposes a model-free MPC planning method which is different from that in [47]. Second, this work has theoretical study for the proposed planner and conducts more experiments in navigation domain. Third, this work limits the application scope from general LTL to LTL in finite traces (LTL$_f$), since the framework in [47] cannot solve every general LTL task which may have infinite task horizon.

Table A3: Hyperparameters of Deep Q Learning in Letter and Room Domain

Hyperparameter	**Value in Letter**	**Value in Room**
Batch size for training options	256	256
Batch size for training planner	64	64
Discount	0.99	0.99
Exploration ϵ init value	0.75	0.75
Exploration ϵ final value	0.05	0.05
Exploration ϵ factor	0.5	0.5
Curriculum level K	5	5
Total number of steps	10e6	10e6
Satisfaction Reward R_F	1	1
Q update interval	10	5
Q target update interval	2000	1500
V update interval	10	5
V target update interval	2000	1500
HER trajectory modification ratio	0.5	1.0
Evaluation interval	10	10
Evaluation episodes	10	10
Optimizer	Adam	Adam
Adam ϵ	2×10^{-5}	2×10^{-5}
β_1, β_2	$0.9, 0.999$	$0.9, 0.999$
Learning rate	3×10^{-4}	2×10^{-4}
Replay buffer size $\|\mathcal{B}\|$	1e6	1e6