Task-Driven Reinforcement Learning With Action Primitives for Long-Horizon Manipulation Skills

Hao Wang, Hao Zhang, Lin Li, Zhen Kan, *Senior Member, IEEE*, and Yongduan Song, *Fellow, IEEE*

***Abstract*—It is an interesting open problem to enable robots to efficiently and effectively learn long-horizon manipulation skills. Motivated to augment robot learning via more effective exploration, this work develops task-driven reinforcement learning with action primitives (TRAPs), a new manipulation skill learning framework that augments standard reinforcement learning algorithms with formal methods and parameterized action space (PAS). In particular, TRAPs uses linear temporal logic (LTL) to specify complex manipulation skills. LTL progression, a semantics-preserving rewriting operation, is then used to decompose the training task at an abstract level, informs the robot about their current task progress, and guides them via reward functions. The PAS, a predefined library of heterogeneous action primitives, further improves the efficiency of robot exploration. We highlight that TRAPs augments the learning of manipulation skills in both learning efficiency and effectiveness (i.e., task constraints). Extensive empirical studies demonstrate that TRAPs outperforms most existing methods.**

***Index Terms*—Action primitives, linear temporal logic (LTL), long-horizon manipulation skills, task-driven RL.**

I. Introduction

ONE OF the ultimate goals of robot learning is to enable robots to evolve like humans via constant interactions with the environment. Although deep reinforcement learning (DRL) has shown great potentials, it, in general, does not perform well in learning long-horizon manipulation skills due to the exploration burden and task constraints. Current research addresses these challenges by performing meaningful exploration. For instance, the robot needs to explore in the action and state space for effective strategies to enable diverse skills (e.g., pick-and-place objects of various sizes and shapes). However, random exploration rarely results in touching the objects, let alone picking them up. Recent works avoid exploration problem via careful engineering to learn manipulation skills [1], [2], [3], while others focus on lowering the exploration burden by exploiting various temporal abstraction frameworks [4], [5], [6]. There are also works on robot learning with task constraints by incorporating formal methods into RL [7], [8], [9]. However, while these methods exhibit improved scalability, they often suffer from prohibitive data efficiency, challenging reward design, weak generalizability, lack of interpretability, and complex task constraints.

One of the major difficulties in RL is that the robot needs to learn both what to do and how to do it in order to accomplish desired tasks. For example, in the Cleanup environment [10] in Fig. 1, the robot needs to learn how to move its end effector to grip or push the object, as well as what to grasp or push and where to move it. Current research has shown that it is not difficult to perform specific motions for a given task, obtains rewards from scalar feedback to adjust the motion execution, and thus completes the given task [11]. However, joint skill learning is challenging for long-horizon manipulation tasks.

An effective solution to the above issues is a hierarchical framework, which decomposes policies into high- and low-level structures. The high-level policies determine what robots need to do in different environment states, while the low-level policies focus on how to do with predefined action primitive modules. A variety of existing works are based on such a hierarchical structure, ranging from task-and-motion planning [11], [12] and neural programming [13], [14], [15] to learning skill libraries based on large offline datasets [16], [17]. While these methods are effective under certain conditions, many of them either rely on explicit domains for planning, involve elaborate reward functions, or require large task-relevant datasets, which limits their scalability. The works of [18] and [19] have augmented DRL algorithms with prebuilt skill modules. These skill modules, which we refer as action primitives, are highly robust and reusable, enabling specific operational goals, such as grasping [20] and collision-free motion planning [21], [22]. Although these methods retain the flexibility of reinforcement learning by using low-level policies as actions for a high-level policy to learn general-purpose behaviors, they are limited due to the nonreconfigurability of rigid primitives, or the composition of hard-coded primitives. The works of [10], [23], and [24] have greatly improved the exploration efficiency and achieved promising results in learning manipulation skills by parameterizing the action space of the DRL. However, they can hardly handle skills with complex logic and temporal constraints.

Recent advances have applied formal methods to RL to learn policies that can satisfy complex task constraints. As a formal language, linear temporal logic (LTL) has been widely

Manuscript received 10 March 2023; revised 17 May 2023 and 23 June 2023; accepted 20 July 2023. Date of publication 11 August 2023; date of current version 23 July 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4701400/4701403, and in part by the National Natural Science Foundation of China under Grant 62173314. This article was recommended by Associate Editor M. Defoort. *(Corresponding author: Zhen Kan.)*

Hao Wang, Hao Zhang, Lin Li, and Zhen Kan are with the Department of Automation, University of Science and Technology of China, Hefei 230026, Anhui, China (e-mail: zkan@ustc.edu.cn).

Yongduan Song is with the School of Automation, Chongqing University, Chongqing 400044, China.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2023.3298195.

Digital Object Identifier 10.1109/TCYB.2023.3298195

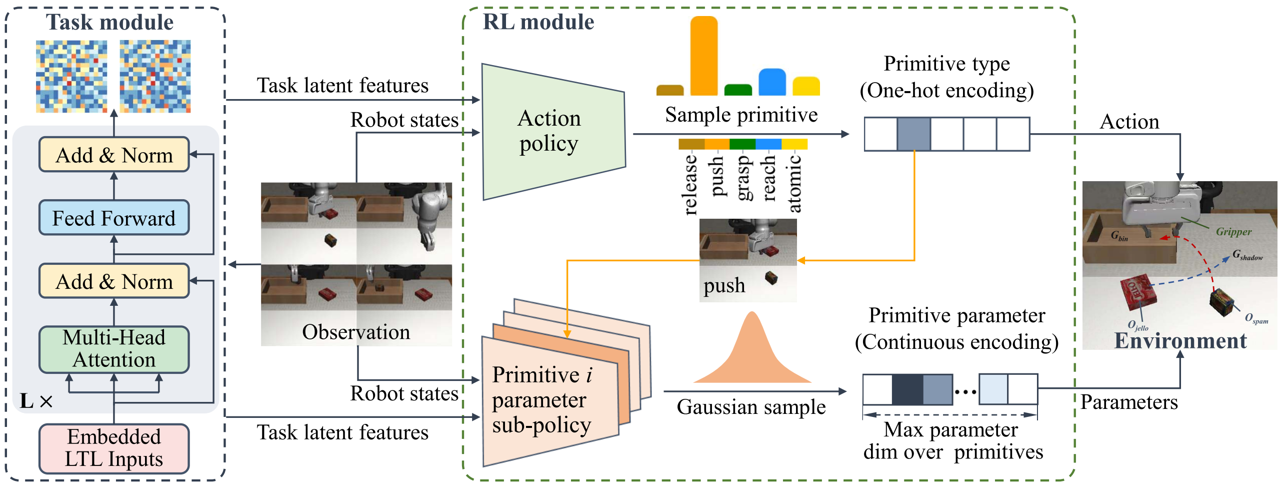


Fig. 1. Framework of TRAPs that enables the robot to leverage action primitives to solve manipulation tasks with temporal logic constraints effectively and efficiently. In the task module, the training manipulation task φ is progressed to the extended task φ_t and then encoded by Transformer as task latent features. In the RL module, the action policy and parameter policies take as input the robot states and task latent features, and then output actions and corresponding parameters to interact with the environment.

used to describe complex logic and temporal constraints due to its rich expressivity and resemblance to natural language [7], [8], [9]. The works of [25], [26], and [27] leverage RL or optimization-based approaches to deal with robotic systems with complex LTL specifications in dynamic and uncertain environments. However, these methods need to construct product automaton, which leads to an exponential increase in computational complexity as the number of states increases, limiting their generalizability [28]. In contrast, the works of [7] and [8] accelerate robot exploration and learning by exploiting LTL progression to generate task instructions. While the computational complexity can be mitigated, few works consider long-horizon manipulation skills due to the heavy exploration burden. Learning from demonstration has been successful for long-horizon manipulation tasks with LTL specifications [29], [30]. However, such methods are limited by the poor generalizability and the high cost of obtaining expert data.

To address the above challenges, in this work, we propose task-driven reinforcement learning with action primitives (TRAPs), a general robot learning framework that uses LTL progression and predefined action primitive libraries to extend RL for long-horizon skills with temporal logic constraints. Particularly, TRAPs is divided into three modules: 1) Env Module; 2) Task Module; and 3) RL Module. The Env Module is an environment-dependent model that preprocesses the observations. In the Task Module, LTL is used to specify complex tasks. LTL progression, a semantics-preserving rewriting operation, is then used to progress the training skill at an abstract level, informs the robot about their current task progress, and guides them via reward functions. To facilitate learning, we design a neat transformer architecture to encode the progressed LTL formula as task latent features at each step. The RL Module is a task-driven hierarchical reinforcement learning framework that takes the robot states and task latent features as input, and then outputs action primitives as well as the parameters for instantiated execution to interact with the environment. Unlike previous works, TRAPs enables the robot to learn task-conditioned policies which can be transferred to new skill variants. Moreover, due to the integration of formal methods and action primitives into the DRL algorithm, TRAPs avoids the complexity of detailed low-level motion planning, while enhancing the generalization, expressivity, and exploration efficiency of DRL. We highlight that TRAPs augments robot learning in accelerating robot learning and solving tasks with temporal logic constraints.

Contributions: The main contributions of this work are summarized as follows.

1) We introduce a task-driven robot learning framework that augments standard reinforcement learning algorithms with LTL progression and action primitives to learn task-conditioned policies and improve exploration efficiency.
2) We construct task-driven labeled PAMDP (TL-PAMDP) based on LTL progression, which overcomes the non-Markovian problem, and theoretically guarantees that the optimal policy of TL-PAMDP with Markovian reward function can satisfy the task specification without compromising the convergence and optimality compared with the optimal policy of labeled PAMDP (L-PAMDP) with non-Markovian reward function.
3) We design a neat transformer architecture that encodes LTL formulas as task latent features to facilitate robot learning with higher expressivity.
4) It is validated via thorough empirical studies that TRAPs significantly outperforms most existing works in terms of learning efficiency and skill performance.

II. Preliminaries and Problem Formulation

A. Co-Safe Linear Temporal Logic

LTL has been widely used to specify complex tasks. Detailed descriptions of the syntax and semantics of LTL can be found in [31]. As a subclass of LTL, Co-safe LTL (sc-LTL) can be fulfilled by finite-length state trajectories [32], which is suitable to describe robotic manipulation skills, for example, pick and place. Hence, we focus on sc-LTL in this work. An LTL formula is co-safe if only temporal operators $\bigcirc$, $\cup$, and $\Diamond$ are used and $\neg$ is only applied to atomic propositions (APs) [33]. Given a set of APs Π, the semantics of an sc-LTL formula is interpreted over a finite sequence $\sigma = \sigma_0\sigma_1\sigma_2 \ldots \sigma_n$, where $\sigma_i \in 2^{\Pi}$. A sequence σ satisfying an sc-LTL φ at time $i \geq 0$ is denoted by $\langle \sigma, i\rangle \models \varphi$.

B. L-PAMDP and Reinforcement Learning

Reinforcement learning algorithms can be leveraged to learn the optimal behavior by interacting with the environment. In general, the environment is modeled as a Markov decision process (MDP). In this work, we consider an MDP with continuous state space $\mathcal{S}$ in $\mathbb{R}^n$ and a parameterized discrete action space $\mathcal{A}$, which is defined as

$$\mathcal{A} = \bigcup_{a\in A_d} \{(a, x)|x \in X_a\} \tag{1}$$

where $A_d = \{a_1, a_2, \ldots, a_k\}$ is a finite set of heterogeneous actions and each $a \in A_d$ corresponds to a set of continuous parameters X_a in $\mathbb{R}^{m_a}$, where m_a is the dimension of X_a. Each action a and its corresponding parameters x form an action tuple $\xi = (a, x)$. Such an MDP is referred to as a parameterized action MDP (PAMDP) [34].

To facilitate the use of LTL representation with robot states, we develop an L-PAMDP by introducing the labeling function L. A labeling function $L : O \to 2^{\Pi}$ is a mapping of an observation to a task state, which can be considered as a collection of event detectors that fires when $p \in \Pi$ holds in the environment and enables the robot to evaluate whether or not an LTL specification is satisfied. For example, in the Pick-and-Place skill, given the observation o that the robot reaches and grasps the soda can, the corresponding task label $L(o)$ is mapped to *can_grasped*, that is, $L(o) = can\_grasped$. The environment observation space O consists of the robot state space S and the object pose S_{obj} in the environment, where $o = (s, s^1_{obj}, s^2_{obj}, \ldots, s^n_{obj}) \in O$ and n indicates the number of objects in the observation space. Note that the object pose is only returned when the agent's exploration reaches the vicinity of the object position. Formally, L-PAMDP is constructed as follows.

Definition 1: An L-PAMDP is a tuple $\mathcal{M}_{PA} = (\mathcal{S}, \mathcal{A}, T, \mathcal{T}, \mathcal{R}_\varphi, \Pi, O, L, \gamma, \mu)$, where $\mathcal{S}$ is the continuous state space, $\mathcal{A}$ is the parameterized discrete action space defined in (1), $T \subseteq \mathcal{S}$ is a finite set of terminal states, $\mathcal{T} = \mathrm{P}(s'|s, (a, x))$ is the transition probability distribution, Π is a set of APs indicating the properties associated with the states, O is the environment observation space, $L : O \to 2^{\Pi}$ is the labeling function, $\gamma \in [0, 1]$ is the discount factor, μ is the initial state distribution, and $\mathcal{R}_\varphi$ is the reward function defined as

$$\mathcal{R}_\varphi = \begin{cases} r_{\text{env}} + \lambda r_{\text{rew}}, & \text{if } \sigma_0\sigma_1\sigma_2, \ldots, \sigma_t \models \varphi \\ r_{\text{env}} - \lambda r_{\text{cost}}, & \text{if } \sigma_0\sigma_1\sigma_2, \ldots, \sigma_t \not\models \varphi \\ r_{\text{env}}, & \text{otherwise} \end{cases} \tag{2}$$

where $\sigma_i = L(o_i)$, r_{env} is the environmental reward, and r_{rew} and r_{cost} correspond to the extra rewards when σ satisfies or does not satisfy the LTL task φ, respectively. The term $\lambda \in (0, 1)$ is a tuning parameter indicating the relative importance between the environmental reward r_{env} and the task reward $r_{\text{rew}}/r_{\text{cost}}$.

The reward function R_φ in Definition 1 indicates that the robot will be given an extra reward r_{rew} if the LTL task φ is completed and will be punished by r_{cost} if it fails. Otherwise, the robot will be awarded by r_{env} based on the task progress. For instance, when performing the Pick-and-Place skill, the robot will get the reward r_{rew} only when the object has been successfully picked and placed and will be punished if the skill fails. If the object has been successfully picked, but not yet be placed, the robot will receive the reward r_{env}.

For a task φ, the robot can interact with the environment following a policy π over an L-PAMDP $\mathcal{M}_{PA}$ to learn the optimal behavior. The policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function that maps a state to an action tuple $\xi = (a, x)$ in the parameterized discrete action space $\mathcal{A}$. Unlike classic RL policies on raw actions, the desired policy π includes a high-level action policy π_a and a set of low-level parameter policies π_p (each heterogeneous action corresponds to a parameter subpolicy), where the action policy π_a determines the type of action primitive $a \in A_d$, and the parameter policy π_p determines the parameters x corresponding to the selected action primitive a. The action primitive a and parameters x together constitute the complete action tuple (a, x) to be performed in state s.

Given the reward function $\mathcal{R}_\varphi$ in (2) and the discount factor $\gamma \in [0, 1]$, the expected discount return under policy π with $s \in \mathcal{S}$ can be defined as

$$U^\pi = \mathbb{E}^\pi \left[\sum_{\infty}^{[t=0]} \Sigma \gamma^t \mathcal{R}_\varphi(s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t, \varphi) \right].$$

The objective of reinforcement learning with L-PAMDP is to search for the optimal policy π^* that maximizes the expected return for each state $s \in \mathcal{S}$, that is, $\pi^* = \operatorname{argmax}_\pi U^\pi(s)$.

In general, the reward function in conventional reinforcement learning is Markovian, that is, the reward at the current state s_t only depends on the transition from the previous state s_{t-1} to s_t. However, when a robot performs a specific task φ, the reward received is closely related to the task completion, that is, $\sigma \models \varphi$, and the episode ends if the task is satisfied or falsified. As mentioned above, the reward function $\mathcal{R}_\varphi(s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t)$ is non-Markovian as the infinite sequence $\sigma = \sigma_0\sigma_1\sigma_2 \ldots$ is defined on the state trajectory $\varsigma = s_0 s_1 s_2 \ldots$ via the labeling function L. In the sequel, we will discuss how to convert non-Markovian rewards into Markovian.

C. Problem Formulation

In this section, we first introduce an example to elaborate on the problems and challenges that need to be addressed in this article, and the example will be used as a running example throughout the work. The problem, along with a mild assumption, is then formally defined.

Example 1: Consider a long-horizon manipulation skill Cleanup in Fig. 1. The robot is required to place the spam can O_{spam} into the storage bin G_{bin} and place the jello box O_{jello} at the upper right shadow corner G_{shadow} with temporal logic constraints. The set of APs Π in this task is

$$\{\text{spam}\_\textit{grasped}, \text{bin}\_\textit{reached}, \text{spam}\_\textit{released}, \text{jello}\_\text{pushed}, \text{shadow}\_\textit{reached}\}.$$

Using the above propositions, cleanup can be formulated by an sc-LTL formula as

$$\varphi_{\text{cleanup}} = \Diamond\left(\varphi_{push\_jellobox} \wedge \Diamond\varphi_{\text{pnp}\_spamcan}\right)$$

where

$$\begin{aligned}\varphi_{push\_jellobox} &= \Diamond(\text{jello}\_\text{pushed} \wedge \Diamond\text{shadow}\_\textit{reached})\\ \varphi_{pnp\_spamcan} &= \Diamond(\text{spam}\_\textit{grasped} \wedge \\ &\qquad \Diamond(\text{bin}\_\textit{reached} \wedge \Diamond\text{spam}\_\textit{released}).\end{aligned}$$

In contrast to the majority of existing works that train the RL policies with hand-designed behaviors for simultaneous execution, the more expressive and parameterized actions are leveraged to train the RL policies in this work. The library of predefined action primitives $\mathcal{A}$ is $\{reach, grasp, push, release, atomic\}$ which is introduced in Section III-A. We emphasize that a wide variety of skills can be solved with this predefined library of action primitives that serve as the building blocks. The LTL instruction is used as task constraints to augment robot learning.

There are two main challenges in Example 1. The first challenge is how to select the optimal action primitive from the library and get its corresponding optimal parameters. Another challenge is how LTL can be used to augment robot learning and make the learned policies satisfy temporal logic constraints. Formally, we first introduce a mild Assumption 1, which is widely employed in the literature, is considered.

Assumption 1: There exists at least one policy $\pi = [\pi_a, \pi_p]$ whose induced path $\varsigma = s_0 s_1 s_2 \ldots$ satisfies the LTL task φ.

Problem 1: Given an sc-LTL task formula φ and an L-PAMDP $\mathcal{M}_{PA} = (\mathcal{S}, \mathcal{A}, T, \mathcal{T}, \mathcal{R}_\varphi, \Pi, O, L, \gamma, \mu)$ corresponding to φ with the reward function $\mathcal{R}_\varphi(s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t)$, the goal is to learn a policy $\pi^* = [\pi_a^*, \pi_p^*]$ that maximizes the discounted rewards, that is, $\pi^* = \text{argmax}_\pi U^\pi(s)$, while satisfying temporal logic constraints and augmenting skill learning via LTL instruction.

III. Methods for Manipulation Skill Learning

This section presents a novel framework, namely, TRAPs, to address Problem 1, which leverages the action primitives to effectively and efficiently learn long-horizon manipulation skills with temporal logic constraints.

A. Building Blocks: Parameterized Action Primitives

The versatile parameterized action primitives serve as the building blocks for diverse manipulation skills. In this work, we consider a primitive library, including five primitives: 1) reach; 2) *grasp*; 3) *push*; 4) *release*; and 5) *atomic*. The first four primitives, as functional APIs, can instantiate action execution via input parameters. It is worth noting that these input parameters with explicit semantics dramatically increase the flexibility and utility when performing complex skills. However, the predefined library of action primitives is hardly universally applicable to diverse environments. To address this problem, additional atomic primitive *atomic* is introduced to fill the gaps that cannot be filled by other action primitives. The following describes the details of each motion primitive [10].

1) *reach:* Move the end effector to a goal position (x, y, z) based on the input 3-D parameters. Up to 20 atomic actions are required for execution.
2) *grasp:* Move the end effector to a pregrasp position (x, y, z) at a yaw angle χ based on the input 4-D parameters, and then close its gripper. Up to 20 atomic actions are required for execution.
3) *push:* The end effector reaches a starting position (x, y, z) at a yaw angle χ and then moves by a displacement $(\delta_x, \delta_y, \delta_z)$ based on the input 7-D parameters. Up to 20 atomic actions are required for execution.
4) *release:* The end effector repeatedly applies atomic actions to open its gripper with no input parameters. Up to 4 atomic actions are required for execution.
5) *atomic:* A single atomic robot action is executed.

These action primitives are implemented by hard-coded closed-loop controllers with different parameters and agnostic to the underlying environment. For example, reach primitive is a function $f_{\text{reach}}(s, x)$ that takes the robot state s and parameters x as input, outputs the target state s^*, and then uses a predefined controller C_{reach} to drive s to s^*. Our RL module, on the other hand, only needs to choose the appropriate primitive with its corresponding parameters based on robot states and task latent features so that the given task with temporal logic constraints can be completed. A key advantage of this decomposition is that the policy can simply focus on what to do instead of learning how to do it, thus speeding up robot learning.

Remark 1: Note that it is difficult to implement a wide variety of manipulation skills through a predefined library of hard-coded action primitives. A possible solution to this challenge is to build a dynamic library of action primitives, that is, continuously enriching the library of action primitives while learning manipulation skills. Since it is beyond the scope of this work that focuses on how to efficiently learn task-conditioned manipulation skills using hard-coded action primitives, this idea will be pursued in our future research.

B. From LTL Progression to Task-Driven RL

Consider an sc-LTL task formula φ and its corresponding L-PAMDP $\mathcal{M}_{PA} = (\mathcal{S}, \mathcal{A}, T, \mathcal{T}, \mathcal{R}_\varphi, \Pi, O, L, \gamma, \mu)$. Our goal is to learn an optimal policy $\pi^* = [\pi_a^*, \pi_p^*]$ while satisfying the temporal logic constraints. To learn such a policy, the robot will be rewarded according to $\mathcal{R}_\varphi(s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t)$, and the episode ends when the task is completed, falsified, or a terminal state is reached.

To solve this problem, a major challenge is that the reward function $\mathcal{R}_\varphi$ depends on the history of states and thus is non-Markovian. Prior works either encode policies through a recurrent neural network (RNN) [35] or develop a reasoning module [36] to address non-Markovian rewards. However, these methods are computationally expensive and might be suboptimal. In this work, we use LTL progression [7], [37] to address the non-Markovian issue. Let $\mathrm{AT}(\varphi)$ denote the proposition required to progress the current LTL formula. The definition of LTL progression is formally given as follows.

Definition 2: Give an LTL formula φ and a truth assignment sequence $\sigma = \sigma_0\sigma_1, \ldots,$ the LTL progression $\mathrm{prog}(\sigma_i, \varphi)$ at step i $\forall i = 0, 1, \ldots,$ is defined as follows:

$$\begin{aligned}
\mathrm{prog}(\sigma_i, p) &= \text{True, if } p \in \sigma_i, \text{ where } p \in \Pi\\
\mathrm{prog}(\sigma_i, p) &= \text{False, if } p \notin \sigma_i, \text{ where } p \in \Pi\\
\mathrm{prog}(\sigma_i, \varphi) &= \begin{cases} \varphi \setminus p, & \text{if } \mathrm{AT}(\varphi) = p, \mathrm{prog}(\sigma_i, p)\\ & = \text{True}\\ \varphi, & \text{otherwise}\end{cases}\\
\mathrm{prog}(\sigma_i, \neg\varphi) &= \neg\mathrm{prog}(\sigma_i, \varphi)\\
\mathrm{prog}(\sigma_i, \varphi_1 \wedge \varphi_2) &= \mathrm{prog}(\sigma_i, \varphi_1) \wedge \mathrm{prog}(\sigma_i, \varphi_2)\\
\mathrm{prog}(\sigma_i, \varphi_1 \vee \varphi_2) &= \mathrm{prog}(\sigma_i, \varphi_1) \vee \mathrm{prog}(\sigma_i, \varphi_2)\\
\mathrm{prog}(\sigma_i, \bigcirc\varphi) &= \varphi\\
\mathrm{prog}(\sigma_i, \varphi_1 \cup \varphi_2) &= \mathrm{prog}(\sigma_i, \varphi_2) \vee (\mathrm{prog}(\sigma_i, \varphi_1) \wedge \varphi_1 \cup \varphi_2).
\end{aligned}$$

After each step in an episode, the operator prog in Definition 2 can be applied to the LTL formula φ that takes the current label σ_i as input and outputs a task specification that reflects which parts of the original task remain to be addressed. For instance, given a manipulation task $\varphi_{\text{cleanup}} = \Diamond(\varphi_{push\_jellobox} \wedge \Diamond\varphi_{pnp\_spamcan})$, it will be progressed to the subsequent task $\widetilde{\varphi}_{\text{cleanup}} = \Diamond\varphi_{pnp\_spamcan}$ once $G_{shadow-}reached$ is completed, and will remain the same otherwise. As discussed in [37], a truth assignment sequence satisfies a given formula at time i if the formula progressed through σ_i is satisfied at time $i+1$. Such a property can be formally presented in the following theorem [37].

Theorem 1: Given any LTL formula φ and the corresponding truth assignment sequence $\sigma = \sigma_i\sigma_{i+1}, \ldots,$ $\langle\sigma, i\rangle \vDash \varphi$ iff $\langle\sigma, i+1\rangle \vDash \mathrm{prog}(\sigma_i, \varphi)$.

According to Theorem 1, using LTL progression in Definition 2 for robot learning has the following benefits. First, since the operator prog is a semantics-preserving rewriting procedure, it can indicate the progression toward task completion with diminished remaining instructions by iteratively applying prog after each step. For instance, the task φ_{cleanup} will progress to $\Diamond\varphi_{pnp\_spamcan}$ once $\Diamond\varphi_{push\_jellobox}$ holds in the environment. With this property, we can design reward functions for robot learning to reflect which parts of the original task formula has been satisfied or unsatisfied yet. Another benefit is its ability to augment robot learning in the following two aspects: 1) accelerating the robot learning since LTL progression can provide more exploration termination conditions while the parameterization of actions can greatly reduce the action space and 2) solving more complex skills with temporal logic constraints.

We apply LTL progression to robot learning by 1) augmenting PAMDP with ongoing LTL task; 2) iteratively applying prog after each step; and 3) additionally rewarding the agent when LTL task progresses to true or false, thus making the reward function $\mathcal{R}_\varphi$ Markovian. Based on the LTL progression in Definition 2, an augmented PAMDP, namely, TL-PAMDP, is developed as follows.

Definition 3 (TL-PAMDP): Given an L-PAMDP without reward function $\mathcal{M}_{PA} = (\mathcal{S}, \mathcal{A}, O, T, \mathcal{T}, \Pi, L, \gamma, \mu)$ corresponding to an LTL task φ, the TL-PAMDP is constructed by $\mathcal{M}_{PA}$ to $\mathcal{M}_{TL} \triangleq (\widetilde{\mathcal{S}}, \mathcal{A}, \widetilde{T}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{R}}_\varphi, \Pi, O, L, \gamma, \widetilde{\mu})$, where $\widetilde{\mathcal{S}} = S \times \mathrm{cl}(\varphi)$, $\widetilde{T} = \{(s, \varphi) | s \in T, \text{or } \varphi \in \{\text{True, False}\}\}$, $\widetilde{\mathcal{T}} = \mathrm{P}((s', \varphi')|(s, \varphi), (a, x)) = \mathrm{P}(s'|s, (a, x))$ if $\varphi' = \mathrm{prog}(L(s), \varphi)$ and $\widetilde{\mathcal{T}} = \mathrm{P}((s', \varphi')|(s, \varphi), (a, x)) = 0$ otherwise, $\widetilde{\mu}(s, \varphi) = \mu(s) \cdot \tau(\varphi)$, $\mathcal{A}$, Π, O, L, and γ are the same as the elements in L-PAMDP, and

$$\widetilde{R}_\varphi = \begin{cases} r_{\text{env}} + \lambda r_{\text{rew}}, & \text{if } \mathrm{prog}(L(s), \varphi) = \text{True}\\ r_{\text{env}} - \lambda r_{\text{cost}}, & \text{if } \mathrm{prog}(L(s), \varphi) = \text{False}\\ r_{\text{env}}, & \text{otherwise.}\end{cases} \tag{3}$$

The term $\mathrm{cl}(\varphi)$ denotes the *progression closure* of φ, that is, the smallest set containing φ that is closed under progression, and $\tau(\varphi)$ denotes the distribution of tasks. If φ is determined, then $\widetilde{\mu}(s, \varphi) = \mu(s)$. The rewards r_{env}, r_{rew}, and r_{cost} are the same as in (2).

With the TL-PAMDP stated in Definition 3, Theorem 2 shows that an optimal policy $\pi^*(\xi|s, \varphi)$ for the TL-PAMDP $\mathcal{M}_{TL}$ yields the same expected discount reward as an optimal policy $\pi^*(\xi_t|s_0, \xi_0, s_1, \xi_1, s_2, \ldots, s_t, \varphi)$ that solves LTL task φ in $\mathcal{M}_{PA}$, where $\pi^* = [\pi_a^*, \pi_p^*]$ includes the optimal action policy and its corresponding optimal parameter policy, and $\xi_i = (a_i, x_i) \in \mathcal{A}$ is a predefined action primitive at stage i.

Theorem 2: Given a TL-PAMDP $\mathcal{M}_{TL} = (\widetilde{\mathcal{S}}, \mathcal{A}, \widetilde{T}, \widetilde{\mathcal{T}}, \widetilde{\mathcal{R}}_\varphi, \Pi, O, L, \gamma, \widetilde{\mu})$ constructed from an L-PAMDP without a reward function $\mathcal{M}_{PA\setminus\mathcal{R}_\varphi} = (\mathcal{S}, \mathcal{A}, T, \mathcal{T}, \Pi, O, L, \gamma, \mu)$ and an LTL task formula φ, an optimal stationary policy $\pi_{TL}^*(\xi|s, \varphi)$ for $\mathcal{M}_{TL}$ can yield the same expected discounted return as an optimal nonstationary policy $\pi_{PA}^*(\xi_t|s_0, \xi_0, s_1, \xi_1, s_2, \ldots, s_t, \varphi)$ for $\mathcal{M}_{PA}$ w.r.t. $\mathcal{R}_\varphi$ for all $s \in S$ and task formula φ.

Proof: Given an LTL formula φ and the corresponding infinite truth assignment sequence $\sigma = \sigma_i\sigma_{i+1}, \ldots,$ where $\sigma_i = L(s_i)$, it holds that $\langle\sigma, i\rangle \vDash \varphi$ iff $\langle\sigma, i+1\rangle \vDash \mathrm{prog}(L(s_i), \varphi)$ according to Theorem 1. Similarly, induction can be used to show that $\sigma_0\sigma_1\sigma_2, \ldots, \sigma_t \vDash \varphi$ iff $\mathrm{prog}(L(s_t), \varphi) = \text{true}$, and thus it is straightforward to use induction to prove that the reward $\mathcal{R}_\varphi$ of $\mathcal{M}_{PA}$ defined in (2) is equal to the reward $\widetilde{\mathcal{R}}_\varphi$ of $\mathcal{M}_{TL}$ defined in (3) at every time step for an LTL task formula φ, initial state $s_0 \in S$, and trajectory $s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t$.

Based on the above discussion, we further discuss the relationship between the optimal policies of $\mathcal{M}_{TL}$ and $\mathcal{M}_{PA}$. Consider any state $s \in S$ and an LTL task formula φ. For a given optimal policy $\pi_{TL}^*(\xi|s, \varphi)$ for $\mathcal{M}_{TL}$, a policy $\pi_{PA}(\xi_t|s_0, \xi_0, s_1, \xi_1, s_2, \ldots, s_t, \varphi)$ for $\mathcal{M}_{PA}$ can be easily constructed by mimicking the action primitives selection of $\pi_{TL}^*(\xi|s, \varphi)$ step by step. This can be done for the following two reasons. First, since the transition probability $\mathrm{P}(s'|s, \xi)$ of reaching state s' is the same for $\mathcal{M}_{TL}$ and $\mathcal{M}_{PA}$ given state s and action primitive ξ, the two policies will induce the same probability distribution over traces. Second, since the reward functions $\widetilde{\mathcal{R}}_\varphi$ and $\mathcal{R}_\varphi$ are equivalent, π_{TL}^* and π_{PA} achieve the same expected discounted return

$$U^{\pi^*_{TL}} = \mathbb{E}^{\pi}\left[\sum_{\infty}^{[i=0]} \Sigma \gamma^{i} \widetilde{\mathcal{R}}_{\varphi}(s_i, \xi_i, s_{i+1}, \varphi)\right].$$

Similarly, if there exists an optimal policy $\pi^*_{PA}(\xi_t|s_0,\ \xi_0,\ s_1,\ \xi_1,\ s_2,\ \ldots,\ s_t,\ \varphi)$ for $\mathcal{M}_{PA}$, we can construct a nonstationary policy $\pi_{TL}(\xi_t|\langle s_0, \varphi\rangle, \xi_0, \langle s_1, \varphi_1\rangle, \xi_1, \ldots, \langle s_t, \varphi_t\rangle)$ for $\mathcal{M}_{TL}$ which mimics the action primitives selection of π^*_{PA} step by step. As discussed above, π^*_{PA} and π_{TL} are able to yield the same expected discounted return

$$U^{\pi^*_{PA}} = \mathbb{E}^{\pi}\left[\sum_{\infty}^{[t=0]} \Sigma \mathcal{R}_{\varphi}(s_0, \xi_0, s_1, \xi_1, s_2, \ldots, \xi_{t-1}, s_t, \varphi)\right].$$

Since $\mathcal{M}_{TL}$ is a TL-PAMDP, a special kind of MDP, there must exist a stationary policy $\pi'_{TL}(\xi|s, \varphi)$ that achieves at least as much return as any nonstationary policy $\pi_{TL}(\xi_t|\langle s_0, \varphi\rangle, \xi_0, \langle s_1, \varphi_1\rangle, \xi_1, \ldots, \langle s_t, \varphi_t\rangle)$. This indicates that $U^{\pi^*_{TL}} \geq U^{\pi'_{TL}} = U^{\pi^*_{PA}}$ Therefore, we show that an optimal stationary policy $\pi^*_{TL}(\xi|s, \varphi)$ for $\mathcal{M}_{TL}$ is as good as an optimal nonstationary policy $\pi^*_{PA}(\xi_t|s_0, \xi_0, s_1, \xi_1, s_2, \ldots, s_t, \varphi)$ for $\mathcal{M}_{PA}$ w.r.t. any $\mathcal{R}_\varphi$, and vice versa. Thus, Theorem 2 is proved. ■

Remark 2: TL-PAMDP is only used for theoretical analyses, rather than constructed in practice. In particular, LTL progression and label functions are used to detect task state, progress the task, and return additional task rewards to guide and facilitate learning. Therefore, the size of the state space explored by the agent does not increase exponentially with the length of the LTL formulas.

Remark 3: Since both $\mathcal{M}_{PA}$ and $\mathcal{M}_{TL}$ are a kind of PAMDP, the construction of policies in the Proof Section III-B is performed in the parameterized action space (PAS), that is, $\mathcal{A} = \bigcup_{a \in A_d}\{(a, x)|x \in X_a\}$, $\xi = (a, x) \in \mathcal{A}$, and the policy $\pi = [\pi_a, \pi_p]$ includes both action policy π_a and parameter policy π_p. These predefined action primitives $\xi \in \mathcal{A}$ are implemented by hard-coded closed-loop controllers with different parameters and agnostic to the underlying environment, so they will not affect the high-level policy learning, which guarantees the correctness of the relevant policies constructed in Proof Section III-B.

Based on Definition 3 and Theorem 2, we convert a non-Markovian decision problem into a Markovian decision problem. In the following, a hierarchical reinforcement learning framework is designed to learn the optimal policy for given TL-PAMDPs.

C. Transformer for LTL Module

LTL formulas can be encoded in a variety of ways, such as RNN [35] and graph neural network (GNN) [8]. However, RNN is not capable of parallel computation and thus computationally inefficient, while GNN struggles to construct generalized graph structures that provide interpretable encoding. In contrast, transformer [38] is capable of modeling global information and has been proven successful in natural language processing. In our previous work [39], a transformer-based encoder for temporal logic, namely, TF-LTL, is developed to encode LTL formulas. TF-LTL has

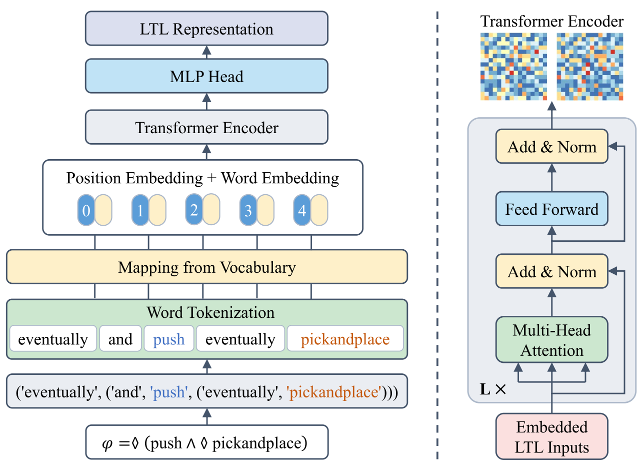


Fig. 2. Overview of the TF-LTL model, which takes as input the LTL formulas and outputs task representations.

two advantages. One is that it enables a better representation of LTL formulas due to its potentials in modeling global information, that is, learning high-quality features by considering the whole context. The other advantage is that it can provide interpretable guidance to the agent, that is, multihead self-attention (MSA) can generate more interpretable models by detecting the distribution of attention in each head.

The overview of the TF-LTL model is shown in Fig. 2, which takes LTL formulas as input and generates task representations. Before fed into the encoder, the given LTL formula φ is first translated into word tokenization $X_\varphi = (x_0, x_1, \ldots, x_n)$, where $x_t, t \in 0, 1, \ldots, n$, represents the operator or proposition of φ, and X_φ is then encoded into word embedding $X_E = [x_0E, x_1E, \ldots, x_nE] \in \mathbb{R}^{B\times L\times D}$, where B is the batch size, L is the length of input X_φ, and D is the model dimension of the TF-LTL. X_E and position embedding PE are summed to obtain a 1-D token embedding sequence as the standard input to the encoder, where the position embedding PE in this work is sine and cosine functions of different frequencies [38]

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\mathrm{pos}/10000^{2i/D}\right)$$
$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\mathrm{pos}/10000^{2i/D}\right)$$

where pos is the position, and i is the dimension. An example is provided in Fig. 2 to show the encoding process.

The structure of the encoder in TF-LTL is shown on the right in Fig. 2, which is composed of identical transformer blocks in a stack. Each transformer block includes an MSA, a position-wise fully connected feedforward (MLP), and a layer norm (LN). The fundamental element in our TF-LTL framework is the MSA, which models global information by considering the whole context to improve the representation of LTL formulas.

Given a 1-D token embedding sequence X from LTL input $X_\varphi = (x_0, x_1, \ldots, x_n)$, it can be linearly transformed to derive the query Q, key K, and value V, which are defined as

$$Q = XW_Q,\quad K = XW_K,\quad V = XW_V$$

where W_Q, W_K, and W_V are linear projection matrices. Then, the similarity between Q and K can be calculated via dot product to obtain attention as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where $\sqrt{d_k}$ is the scaling factor. The global computation procedure of TF-LTL can be represented as follows:

$$\begin{aligned}
X_0 &= [x_0E; x_1E; \cdots ; x_nE] + E_{\text{pos}}, \ \ E_{\text{pos}} \in \mathbb{R}^{L\times D}\\
\widehat{X}_l &= \text{MSA}\big(\text{LN}(\widehat{X}_l)\big) + X_{l-1}, \ \ l = 1, 2, \ldots, L\\
X_l &= \text{MLP}\big(\text{LN}(\widehat{X}_l)\big) + \widehat{X}_l, \ \ l = 1, 2, \ldots, L\\
X_{\text{out}} &= \text{LN}(X_l)
\end{aligned}$$

where X_{out} denotes the task latent feature output from TF-LTL, which can be manually designed with suitable dimension for particular tasks.

It might be arduous to train both TF-LTL and RL modules in TRAPs at the same time. Therefore, similar to our previous work [39], we propose an environment-agnostic pretraining scheme on the basis of TL-PAMDP. The scheme includes: 1) constructing a special TL-PAMDP $\mathcal{M}^s_{TL}$ with an LTL task φ and a single-state L-PAMDP without reward $\mathcal{M}^s_{PA} = (\mathcal{S}, \mathcal{A}, T, \mathcal{T}, \Pi, O, L, \gamma, \mu)$, where $\mathcal{S} = \{s_0\}$, $T = \emptyset$, $\mathcal{A} = \Pi$, $\mathcal{T} = \text{P}\{s_0|s_0, .\} = 1$, $\mu(s_0) = 1$, $L(s_0, p) = \{p\}|p \in \Pi$; 2) training TF-LTL to convergence on TL-PAMDP $\mathcal{M}^s_{TL}$; and 3) transferring the learned TF-LTL as the initial LTL module to the downstream $\mathcal{M}_{TL}$.

This pretraining scheme can be considered as solving a special kind of TL-PAMDP. Since the pretraining scheme exploits the environment-agnostic nature of LTL semantics, it is more efficient than directly training complete models in the downstream environment. Moreover, due to the modular structure of TRAPs, the task module is robust to the environment as long as the task formula remains constant.

Overall, given a target TL-PAMDP, our goal is to learn useful encodings for the corresponding task formula and then uses these encodings to solve the TL-PAMDP. The advantage of our approach over traditional methods is that, as the TF-LTL model guides the robot to better interact with the environment, it is also gradually updated during the interaction, further facilitating the robot's understanding of the current subtask. This leads to a mutual improvement, that is, the TF-LTL guides the robot's actions, and the results of the actions improve the TF-LTL for better task instructions.

D. Algorithm Architecture

As shown in Fig. 1, the framework of TRAPs consists of three modules.

1) *Env Module:* An environment-dependent model that preprocesses the observations. In this work, the desired environment observations are obtained directly from embedded functions in Robosuite [40].
2) *Task Module:* A transformer model for the LTL task formula (discussed in Section III-C).
3) *RL Module:* A task-driven hierarchical reinforcement learning framework which decides action primitives and its corresponding parameters to take in the environment. In principle, the hierarchical policy framework can be integrated with any DRL algorithm designed for continuous control. In this article, soft actor–critic (SAC) [40], a state-of-the-art DRL algorithm, is applied due to its excellent performance.

For implementation, TRAPs takes as input the observations of the environment and the current task encoded by the task module and outputs an action primitive and its corresponding parameters to interact with the environment. The hierarchical framework from [10] is adopted, in which the RL module contains a high-level action policy and a low-level parameter policy. The action policy is represented as a single neural network, and the parameter policy is a collection of subnetworks, where each subnetwork corresponds to an action primitive. This structured framework allows us to accommodate primitives with heterogeneous parameters. These parameter policy subnetworks are designed to allow batch tensor computation for primitives with different parameter dimensions, and all output a "one-size-fits-all" distribution for the parameters $x \in \mathbb{R}^{d_A}$, where $d_A = \max_a d_a$ is the maximum parameter dimension over all primitives. During primitive execution, the parameter x will be simply truncated to the length d_a of the selected primitive a. The visual illustration of TRAPs is shown in Fig. 1.

While our policy framework is irrelevant to the choice of the DRL algorithm, in our experiments, we opted for SAC to maximize environment rewards as well as the policy entropy. We adapt SAC by modifying its standard critic neural network $Q_\theta(s, a)$ and actor neural network $\pi_\phi(a|s)$ to $Q_\theta(s, \xi, \varphi)$ and the hierarchical policy networks $\pi_{a_\phi}(a|s, \varphi)$ and $\pi_{p_\psi}(x|s, a, \varphi)$. The loss for the critic, action policy, and parameter policy are then designed, respectively, as

$$\begin{aligned}
J_Q &= \Big(Q_\theta(s, \xi, \varphi) - \Big(r(s, \xi, \varphi) + \gamma\Big(Q_{\bar{\theta}}\Big(s', \xi', \varphi'\Big)\\
&\quad - \alpha_a\log\big(\pi_{a_\phi}\big(a'|s', \varphi'\big)\big)\\
&\quad -\alpha_p\log\big(\pi_{p_\psi}\big(x'|s', a', \varphi'\big)\big)\Big)\Big)\Big)^2\\
J_{\pi_a}(\phi) &= \mathop{E}_{a\sim\pi_{a_\phi}}\Big[\alpha_a\log\big(\pi_{a_\phi}(a|s, \varphi)\big)\\
&\quad - \mathop{E}_{x\sim\pi_{p_\psi}}[Q_\theta(s, \xi, \varphi)]\Big]\\
J_{\pi_p}(\psi) &= \mathop{E}_{a\sim\pi_{a_\phi}}\mathop{E}_{x\sim\pi_{p_\psi}}\Big[\alpha_p\log\big(\pi_{p_\psi}(x|s, a, \varphi)\big)\\
&\quad -Q_\theta(s, \xi, \varphi)\Big].
\end{aligned}$$

Here, $r(s, \xi, \varphi) \in \widetilde{R}_\varphi$, α_a, and α_p determine the maximum entropy for the action policy and parameter policy, respectively. Since φ and φ' in J_Q are encoded by TF-LTL, the weights of TF-LTL can be updated by backpropagation of J_Q.

While TRAPs can accelerate exploration, reasoning with temporal expanded actions remains to present exploration challenges [41]. In this article, we address this issue using the approach in [10] that equips robots with affordances to help discern the utility of actions in different contexts, for example, grasping is only performed when the graspable objects are close, while pushing is only performed when the pushable objects are close. For implementation, an auxiliary affordance score $s_{\text{aff}}(s, x; a) \in [0, 1]$ is added to the reward function $r_t \in \widetilde{R}_\varphi$ to express these affordances, which measures the affinity of the parameter x for a given primitive a in a particular state s. Concretely, to ensure the universal applicability of *atomic* and *release* primitives, the affordance score is set to 1, while reach, *grasp*, and *push* are designed to encourage the robot to reach the relevant area of interest in the workspace as

$$s_{\text{aff}}(s, x; a) = \max_{p\in P} 1 - \tanh(\max((\|x_{\text{reach}} - p\| - \epsilon), 0))$$

where x_{reach} is the target position of action primitives, ϵ is a threshold, and $p \in P$ is a set of key points, for example, the key points p for pushing are the locations of objects to push.

The pseudocode of TRAPs is outlined in Algorithm 1. In the exploration phase, we progress the given task φ so that it takes into account the sequence of states seen thus far on each step of any episode (line 9). If the current task satisfies $\varphi_t \in \{\text{True}, \text{False}\}$ or the robot state $s \in T$, we end the current episode for the next phase of exploration (lines 10 and 11). Otherwise, φ_t is encoded as task latent features by TF-LTL as introduced in Section III-C (line 13). The primitive type a_t and its corresponding parameters x_t are sampled from action policy π_{a_ϕ} and parameter policy π_{p_ψ} and then executed in environment to obtain reward r_t and next state s_{t+1} (lines 15–18). To facilitate exploration, an additional reward based on the affordance score is added to the reward function (line 19). The transitions are stored in the replay buffer (line 20). In the training phase, the Q network, TF-LTL, action policy, and parameter policy are updated via gradient descent, respectively, on each step (lines 25–28).

Algorithm 1 TRAPs

```
1: procedure INPUT: (An LTL formula $\varphi$ and the TL-PAMDP $\mathcal{M}_{TL}$ corresponding to $\varphi$)
       Output: Optimal action policy $\pi^*_{a_\phi}(a|s, \varphi)$ and its corresponding optimal parameter policy $\pi^*_{p_\psi}(x|s, a, \varphi)$
       Initialization: Q network $Q_\theta(s, \xi, \varphi)$, action policy $\pi_{a_\phi}(a|s, \varphi)$, parameter policy $\pi_{p_\psi}(x|s, a, \varphi)$, replay buffer $\mathcal{D}$
2:     Load the pretrained weights $\chi$ to the TF-LTL module
3:     for iteration 1,2,...,N do
4:         for episode 1,2,...,M do {Exploration Phase}
5:             Initialize timer $t \leftarrow 0$
6:             Initialize episode $s_0$
7:             Initialize LTL formula $\varphi_0 \leftarrow \varphi$
8:             while episode not terminated do
9:                 $\varphi_t \leftarrow \text{prog}(L(s), \varphi_{t-1})$
10:                if $\varphi_t \in \{\text{True, False}\}$ or $s \in T$ then
11:                    Break
12:                else
13:                    $\varphi_t$ is encoded by TF-LTL as $\varphi_{em}$
14:                end if
15:                Sample primitive type $a_t$ from action policy $\pi_{a_\phi}(a_t|s_t, \varphi_{em})$
16:                Sample primitive parameters $x_t$ from parameter policy $\pi_{p_\psi}(x_t|s_t, a_t, \varphi_{em})$
17:                Truncate sampled parameters to dimension of sampled primitive $x_t \leftarrow x_t[:d_{a_t}]$
18:                Execute $a_t$ and $x_t$ in environment, obtain reward $r_t$ and next state $s_{t+1}$
19:                Add affordance score to reward $r_t \leftarrow r_t + \lambda s_{aff}(s_t, x_t; a_t)$
20:                Add transition to replay buffer $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t, x_t, r_t, \varphi_t, s_{t+1}\}$
21:                Update timer $t \leftarrow t + 1$
22:            end while
23:        end for
24:        for training step 1,2,...,K do {Training Phase}
25:            Update Q network: $\theta \leftarrow \theta - \lambda_{lr}\nabla_\theta J_Q$
26:            Update TF-LTL: $\chi \leftarrow \chi - \lambda_{lr}\nabla_\chi J_Q$
27:            Update action policy: $\phi \leftarrow \phi - \lambda_{lr}\nabla_\phi J_{\pi_a}(\phi)$
28:            Update parameter policy: $\psi \leftarrow \psi - \lambda_{lr}\nabla_\psi J_{\pi_p}(\psi)$
29:        end for
30:    end for
31: end procedure
```

E. Theoretical Advantages of TRAPs

In contrast to conventional DRL algorithms, the action space of TRAPs consists of heterogeneous parameterized action primitives, that is, $\mathcal{A} = \bigcup_{a\in A_d}\{(a, x)|x \in X_a\}$. These action primitives are agent-centric as their execution is implemented by the underlying hard-coding and depends only on the robot states. The RL policy of TRAPs is environment-centric, which selects appropriate action primitives as well as their corresponding parameters to instantiate its execution based on the current environment state to accomplish the desired task. This PAS-based RL framework offers several advantages. First, it allows the robot to focus on learning what to do rather than on learning how to do it, which greatly reduces the learning and exploration burden. Also, since each action in the PAS has a clear physical significance, such as *reach, push, grasp,* etc., making RL policies more interpretable. That is, given a set of n viable policies for a task $\mathcal{T} : \{\tau^i\}_{i=1}^n = \{(s_1^i, \varphi_1^i), (a_1^i, x_1^i), \dots, (a_{T_i}^i, x_{T_i}^i), (s_{T_i+1}^i, \varphi_{T_i+1}^i)\}_{i=1}^n$, the corresponding action sketches, which capture the high-level task semantics and provide useful abstractions, can be represented as $\{K^i\}_{i=1}^n = \{a_1^i, a_2^i, \dots, a_{T_i}^i\}_{i=1}^n$, and, from these action sketches, we can easily understand the internal structure of the RL policies. For instance, one of the action sketches for Pick-and-Place skill is {grasp, reach, release}, which means that the robot needs to pick up a soda can, and moves it to a specific position, then releases the soda can. This interpretability provides a basis for policy transferability, that is, semantically similar skills have similar structures of action policies with different parameters. By taking advantage of the hierarchical structure of TRAPs, we can reuse the action policies from semantically similar skills or expert demonstrations and only need to update the parameter policies for new skills, thus significantly accelerating robot learning. The above advantages guarantee promising performance of our method by experiments as shown in Section IV.

In addition, by integrating with formal methods, TRAPs is capable of decomposing the long-horizon skill at an abstraction level via LTL progression, informing the robot their current task progress, and guiding robot learning via reward functions. This task-driven method provides additional termination conditions for the learning process to improve the exploration efficiency, that is, the current training episode ends when the task is completed, falsified, or a terminal state is reached. Meanwhile, as discussed before, the reward function $\mathcal{R}_\varphi$ is non-Markovian when learning policies that satisfy LTL task constraints on the L-PAMDP. To overcome this issue, TRAPs constructs a novel TL-PAMDP with Markovian reward function $\widetilde{R}_\varphi$ by using LTL progression, which theoretically ensures that the convergence and optimality are not compromised. We demonstrate in Section IV the advantages of using task-driven approaches for diverse long-horizon manipulation skills. Moreover, a TF-LTL is developed to encode LTL formulas in this work. Benefiting from the capability of Transformer in modeling global information, that is, learning high-quality

TABLE I
SKILLS, SKILL DESCRIPTIONS, AND CORRESPONDING LTL FORMULAS

Skills	Skill descriptions	LTL Formulas
Lift	Pick up a cube and lift it above the table.	$\varphi_{lift} = \Diamond (cube\_grasped \wedge \Diamond cube\_lifted)$
Door Opening	Turn the door handle and open the door.	$\varphi_{door} = \Diamond (handle\_grasped \wedge \Diamond door\_opened)$
Pick and Place	Pick up a soda can and place it into a specific target compartment.	$\varphi_{pnp} = \Diamond (can\_grasped \wedge \Diamond (target\_reached \wedge \Diamond can\_released))$
Stack	Stack a cube on top of another cube.	$\varphi_{stack} = \Diamond (cube\_grasped \wedge \Diamond (top\_reached \wedge \Diamond cube\_released))$
Nut Assembly	Fit a nut tool onto the round peg.	$\varphi_{nut} = \Diamond (nut\_grasped \wedge \Diamond (roundpeg\_reached \wedge \Diamond nut\_released))$
Cleanup	Push a jello box at the upper right corner and then store a spam can into a bin.	$\varphi_{cleanup} = \Diamond \left(\varphi_{push\_jellobox} \wedge \Diamond \varphi_{pnp\_spamcan}\right)$

features by considering the whole context, TF-LTL is able to better represent LTL formulas. Also, TF-LTL can provide improved interpretable instruction for agents, as multiheaded self-attentiveness is able to produce more interpretable models by the attention distribution of each head. We demonstrate the promising performance of transformer-based encoders for LTL formulas in Section IV.

IV. EXPERIMENTS

In this section, we evaluate TRAPs against previous work. Particularly, extensive experiments are carried out to investigate 1) *Performance:* whether TRAPs outperforms previous methods in terms of learning efficiency and effectiveness; 2) *Expressivity:* whether TRAPs can facilitate the understanding of the LTL task via TF-LTL, and thus better guide the selection and combination of action primitives to complete the task; 3) *Satisfiability:* whether TRAPs can synthesize task-conditioned policies to satisfy task specifications; and 4) *Transferability:* whether TRAPs can be applied to semantically similar skill variants to improve the learning efficiency.

A. Experimental Setup

Environments and Skills: Robosuite [42], a framework for manipulation tasks that emphasizes realistic simulation and control, is employed in this work to evaluate the performance of TRAPs. For the purpose of comparison, six manipulation skills of different complexities in [10] are selected. The detailed descriptions and the corresponding LTL formulas of the skills are presented in Table I. At each step, the robot will select and execute either an atomic action or a nonatomic action primitive with specific parameters as outlined in Section III-A, and return 1) a Markovian reward signal for robot learning; 2) the observations consisting of the robot's state and the object's pose in the environment; and 3) the progressed LTL formula that indicates the progress toward task completion. The Markovian reward function used for the experiments is shown in (3), where r_{env} is the default reward setting in Robosuite [42], $r_{\text{rew}} = 10$, $r_{\text{cost}} = 10^{-5}$, and $\lambda = 0.75$. All evaluations are implemented on a desktop running Ubuntu 18.04 with Intel Core i9 CPU and NVIDIA 3090 GPU.

Baselines: The first (also the simplest) baseline is the standard SAC model [40], which executes only atomic primitive. One way to improve the efficiency of reinforcement learning and synthesize task-conditioned policies is to generate task instructions using LTL semantics to guide robot learning [8]. In this article, we implement this method to learn manipulation skills by extending the standard SAC with GNN-encoded LTL semantics (**SAC$_{\textbf{GNN-LTL}}$**) and using it as a baseline for our method. Another important baseline is the manipulation primitive-augmented reinforcement learning (MAPLE) [10], which is an augmented reinforcement learning framework based on a library of predefined behavioral primitives. To demonstrate the superiority of Transformer-encoded LTL semantics in robot learning, thanks to the modularity of our approach, we develop two additional baselines by modifying the task modules in TRAPs: one is based on GNN-encoded LTL (**TRAPs$_{\textbf{GNN-LTL}}$**) and the other is encoded with Transformer (**TRAPs$_{\textbf{TF-LTL}}$**). In addition, when comparing the success rate of skills, besides the baselines described above, we add three additional baselines, including: 1) double actor–critic (DAC) [43]; 2) open-loop task policy (**Open Loop**) [44]; and 3) hierarchical reinforcement learning that determines the primitive type and parameters independently (**Flat**) [45].

B. Main Experimental Results

In this section, TRAPs is evaluated in terms of performance, expressivity, satisfiability, and transferability.

1) Performance: TRAPs and the baseline approaches are employed to perform the six manipulation skills. Fig. 3 shows the evolution of reward for the three main baselines and the two variants of TRAPs during training. The total training times are listed in Table II, where \ indicates the failure of skill learning. It is observed that 1) algorithms with action primitives (MAPLE, TRAPs$_{\text{GNN-LTL}}$, and TRAPs$_{\text{TF-LTL}}$) perform better than those without action primitives (SAC and SAC$_{\text{GNN-LTL}}$), especially when relatively complex skills (Pick and Place, Stack, Nut Assembly, and Cleanup) are considered, that is, receiving 2–3 times higher rewards; 2) TRAPs can achieve the same performance as MAPLE for relatively simple skills (Lift, Door Opening, Pick and Place, and Stack) and shows better performance in complex skills (Nut Assembly and Cleanup); 3) TRAPs shows higher learning efficiency than MAPLE and such an advantage becomes more significant with the increase of skill complexity, since LTL progression can decompose the training skill at an abstract level, informs the robot about their current task progress, and guides them to complete tasks via reward functions as discussed in

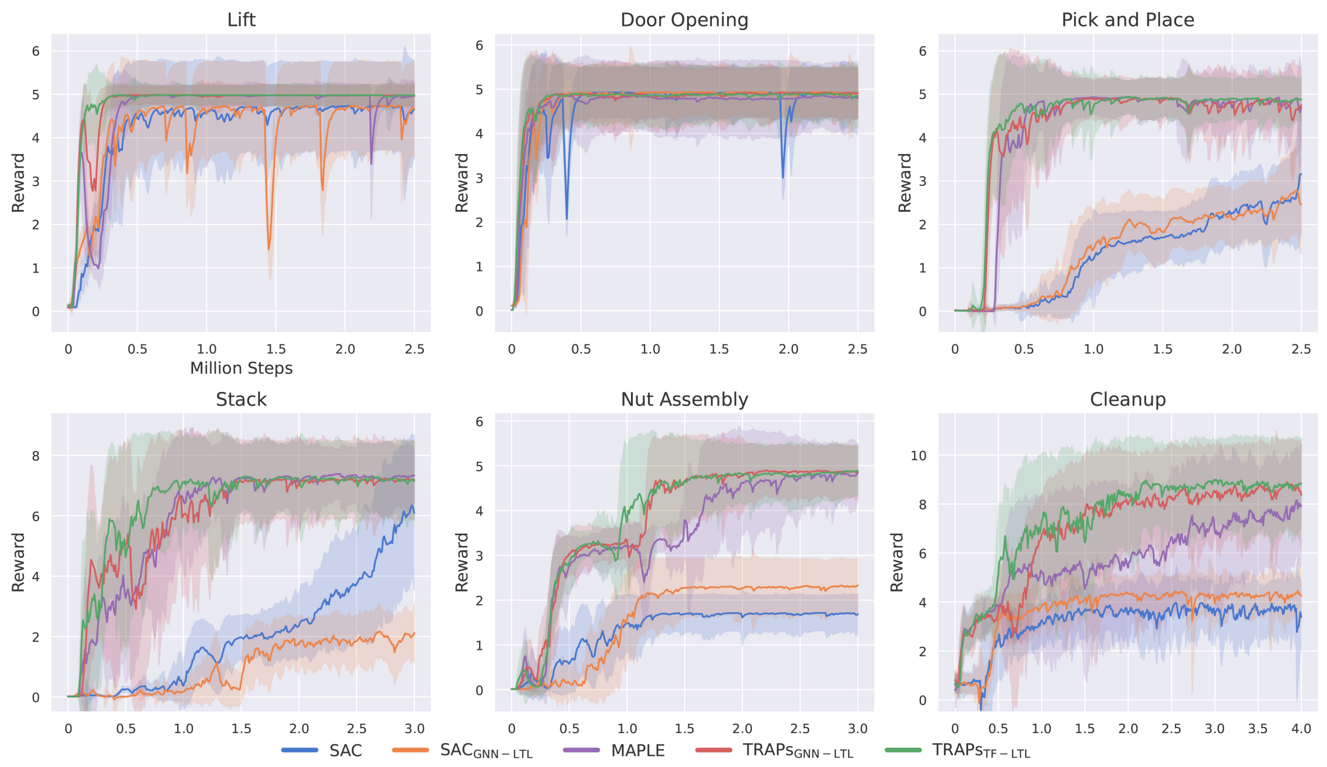


Fig. 3. Plots of normalized reward curves. These learning curves show the mean and standard deviation of the episodic task reward throughout training. All experiments are averaged over six seeds.

TABLE II
TOTAL TRAINING TIME (HOURS)

Action Primitives	Lift	Door Opening	Pick and Place	Stack	Nut Assembly	Cleanup
SAC [40]	111.20	97.85	\	\	\	\
SAC$_{\text{GNN-LTL}}$ [8]	121.20	107.16	\	\	\	\
MAPLE [10]	118.83	108.96	128.74	128.52	158.89	157.20
TRAPs$_{\text{GNN-LTL}}$	112.44	107.30	122.79	127.00	155.67	145.54
TRAPs$_{\text{TF-LTL}}$	**101.85**	**94.49**	**112.83**	**114.64**	**146.62**	**126.50**

Section III-B; 4) TRAPs$_{\text{TF-LTL}}$ always performs similarly or better than TRAPs$_{\text{GNN-LTL}}$ and shows higher learning efficiency, especially in complex skills (Cleanup). This is due to the two advantages of Transformer: a) the ability to better represent the LTL formula and b) the interpretable instructions to agents as stated in Section III-C; and 5) TRAPs$_{\text{TF-LTL}}$ uses the shortest training time compared to all baselines, with over 12% reduction against the method without LTL (MAPLE).

The success criteria for skill completion from [10] are employed to further evaluate the performance of TRAPs and all baselines. The trained policy is evaluated by 20 episodes with the mean of their success rates as our final skill success rate. The results of the success rate are listed in Table III. First, it is observed that TRAPs$_{\text{TF-LTL}}$ achieves the highest success rate compared to all baselines, that is, 100% in most of the evaluation skills except Cleanup. Although the success rate of MAPLE is close to ours, TRAPs has a slightly better success rate than MAPLE overall. As discussed before, due to the guidance provided by LTL in the learning process, TRAPs has better learning efficiency. Second, algorithms with action primitives (MAPLE, TRAPs$_{\text{GNN-LTL}}$, and TRAPs$_{\text{TF-LTL}}$) show a much higher success rate than algorithms (DAC, SAC, and SAC$_{\text{GNN-LTL}}$) without action primitives. Especially when considering complex manipulation skills (e.g., Nut Assembly and Cleanup), algorithms without action primitives are hard to succeed due to the exploration burden and task constraints. While the Open-Loop baseline is capable of solving basic skills, such as Door Opening and Stack, it struggles when the skill requires the robot to perform adaptive reasoning on the current state. The Flat baseline uses a hierarchical framework with action primitives; however, it still struggles to perform all skills, especially when the skills are complex. In contrast, the hierarchical structure in TRAPs is the key to proper reasoning over a library of heterogeneous action primitives. Overall, the results show that TRAPs that integrates RL with formal methods and PAS can efficiently learn diverse manipulation skills.

2) Expressivity: The expressivity of TRAPs is evaluated in two ways: 1) the internal structural composability of manipulation task policies and 2) the advantages of encoding LTL formulas with TF-LTL. We first introduce the compositionality score from [10], a quantifiable metric to measure the degree of compositional behavior within a learned policy.

TABLE III
FINAL SKILL SUCCESS RATE (%)

Action Primitives	Lift	Door Opening	Pick and Place	Stack	Nut Assembly	Cleanup
DAC [43]	75.0 ± 12.7	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
SAC [40]	98.0 ± 2.4	98.0 ± 1.3	0.0 ± 0.0	38.0 ± 28.7	0.0 ± 0.0	0.0 ± 0.0
$\text{SAC}_{\text{GNN-LTL}}$ [8]	92.0 ± 2.6	100.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Open Loop [44]	43.0 ± 43.5	100.0 ± 0.0	81.0 ± 38.0	85.0 ± 3.2	0.0 ± 0.0	0.0 ± 0.0
Flat [45]	61.0 ± 47.8	100.0 ± 0.0	1.0 ± 2.0	98.0 ± 2.4	0.0 ± 0.0	0.0 ± 0.0
MAPLE [10]	100.0 ± 0.0	100.0 ± 0.0	95.0 ± 7.7	98.0 ± 2.4	99.0 ± 2.0	91.0 ± 5.8
$\text{TRAPs}_{\text{GNN-LTL}}$	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	97.5 ± 3.5	100.0 ± 0.0	93.0 ± 2.2
$\text{TRAPs}_{\text{TF-LTL}}$	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**95.0 ± 2.1**

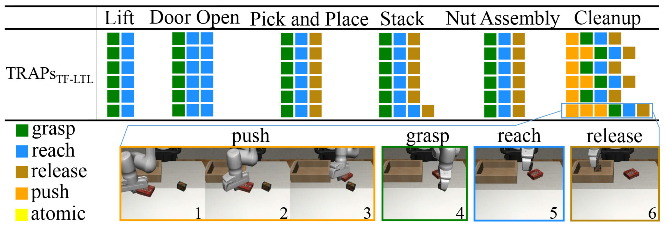


Fig. 4. (Top) Visualization of action sketches corresponding to the learned policies of the agent utilizing $\text{TRAPs}_{\text{TF-LTL}}$ in six evaluation environments. Each row corresponds to a single sketch progressing temporally from left to right. (Bottom) Visualization of action sketches and snapshots for φ_{cleanup}.

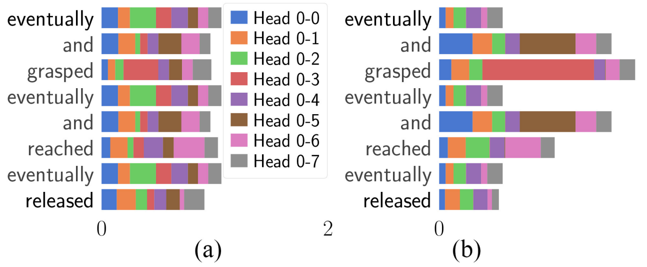


Fig. 6. Heads concentration from attention view of TF-LTL on φ_{pnp}. The weights before and after the convergence of MSA for the TF-LTL zero layer correspond to (a) and (b), respectively.

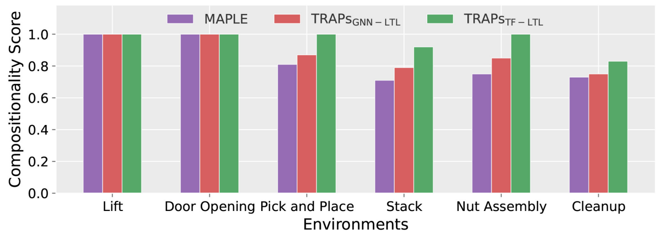


Fig. 5. Visual representation of composability scores. TRAPs has higher compositionality scores than MAPLE. Among the two variants of TRAPs, $\text{TRAPs}_{\text{TF-LTL}}$ has higher compositionality scores than $\text{TRAPs}_{\text{GNN-LTL}}$.

Given a set of action sketches $\{K^i\}_{i=1}^n = \{a_1^i, a_2^i, \ldots, a_{T_i}^i\}_{i=1}^n$, the compositionality score can be defined as

$$f_{\text{comp}}(\mathcal{T}, \mathcal{L}) = \frac{1}{n(n-1)} \sum_{i \neq j} \left(1 - \frac{d_{\text{Lev}}(K_i, K_j)}{\max(|K_i|, |K_j|)}\right) \quad (4)$$

where $d_{\text{Lev}}(K_i, K_j)$ is the Levenshtein distance [46] among action sketches and $\mathcal{L}$ is the library of action primitives. Note that each nonatomic primitive type is treated as a unique token and each individual occurrence of an atomic primitive is also treated as a unique token in this work. A higher score means better compositionality.

The action sketches of $\text{TRAPs}_{\text{TF-LTL}}$ with six seeds are visualized in Fig. 4. The compositionality scores are visualized in Fig. 5. Apparently, given a library of action primitives, TRAPs can select and combine appropriate action primitives from the library to complete diverse skills. Moreover, from the action sketch, user can conveniently observe the rationality of the action primitive selection based on the skill. Fig. 5 also indicates that TRAPs has higher compositionality scores than MAPLE, since the use of LTL can facilitate the robot's selection of action primitives by decomposing the task into subtasks. Among the two variants of TRAPs, $\text{TRAPs}_{\text{TF-LTL}}$ has higher compositionality scores than $\text{TRAPs}_{\text{GNN-LTL}}$, since Transformer can better represent the LTL formula and provides interpretable instructions in learning.

To illustrate the expressivity of TRAPs, take the Pick-and-Place skill for instance. The weights before and after the convergence of MSA for the TF-LTL zero layer are shown in Fig. 6, in which the bars of different colors and lengths represent different heads and their weights on the corresponding token. The weights of the tokens (APs) are almost the same at the start of training as shown in Fig. 6(a), which indicates that the robot has no clear preference for the current LTL instruction. When the training is over, the weight of "grasped" is significantly greater than the other tokens, indicating that there is a higher chance that grasped will be selected, as shown in Fig. 6(b). It further demonstrates the rationality and expressivity of our framework in selecting and composing action primitives into policies.

3) Satisfiability: We verify the task satisfiability of TRAPs using the Cleanup skill environment, that is, whether TRAPs can synthesize task-conditioned policies. We define a task constraint $\varphi'_{\text{cleanup}} = \lozenge(\varphi_{pnp\_spamcan} \wedge \lozenge\varphi_{push\_jellobox})$ in an opposite order of φ_{cleanup}. In English, $\varphi'_{\text{cleanup}}$ means "the robot must first place the spam can spam into the storage bin *bin*, and then places the jello box jello at the upper right shadow corner shadow." In Figs. 4 and 7, the action sketches and snapshots corresponding to the policies with φ_{cleanup} and $\varphi'_{\text{cleanup}}$ constraints are visualized, respectively. It can be seen that, given diverse task constraints, our method is able to synthesize the corresponding task-conditioned policies to successfully perform these tasks via adjusting the LTL formulas in the task

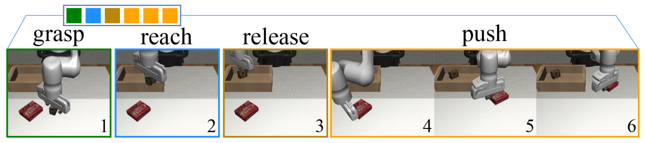


Fig. 7. Action sketches and snapshots for $\varphi'_{\text{cleanup}}$.

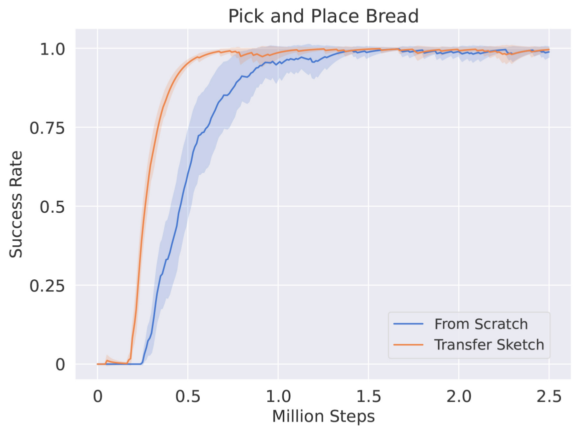
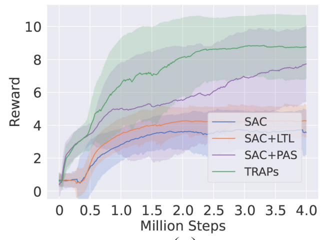


Fig. 8. Success rate curves for learning from scratch and transfer-based policy learning in the Pick-and-Place Bread skill.



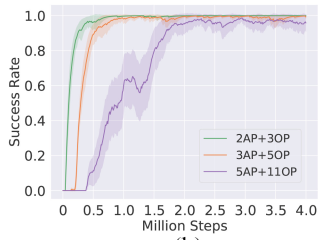


Fig. 9. Ablation experiment results. (a) Reward curves for ablation experiments on the role of LTL and PAS in the Cleanup skill. (b) Success rate curves for skill learning with different lengths of LTL formula specification.

module without modifying other parts of the algorithm. As TRAPs is modular, the above-mentioned properties grant more flexibility and robustness to our algorithm when addressing diverse skills.

4) Transferability: The expressivity of TRAPs provides a basis for policy transfer to similar skills. Since semantically similar skills generally have similar structures of action sketches or action policies but different parameters, when encountering a new semantically similar skill, we only need to update the parameter policy. This idea is demonstrated using a Pick-and-Place environment, where the action policy of picking and placing a soda can is transferred to a similar skill, but with a different object (e.g., bread) and different target placement. The success rate curves for learning from scratch and transfer-based policy learning are presented in Fig. 8. It is observed that the transfer-based policy is over three times as efficient as learning from scratch. The experimental results illustrate that the learned action sketches or action policies can be reused to semantically similar skills for rapid application to related skill variants.

C. Ablation Experiments

Ablation experiments are conducted to show how LTL and PASs contribute to the performance of TRAPs. In particular, for the learning of Cleanup skill, TRAPs is compared with a set of variants: 1) without LTL and PAS (SAC); 2) without PAS (SAC+LTL); and 3) without LTL (SAC+PAS). Fig. 9(a) shows the reward curves, which indicate that both LTL and PAS are able to improve the performance of manipulation skill learning, and PAS alone outperformed LTL alone. Our approach (TRAPs) achieves the best performance compared to the first two. In addition, the use of LTL enforces the satisfaction of the task specification, as described in Section IV-B.

Since the length of an LTL formula is determined by the number of APs and operators (OP), experiments are conducted to show how the performance of the learned policies varies with the number of APs and OPs in the LTL formula. Fig. 9(b) demonstrates the success rate of skill learning with different length of LTL formula specification. It can be seen that as the length of the LTL formula increases, that is, with more task constraints, the efficiency of skill learning decreases, and the performance of the learned policies drops slightly. However, they are still in an acceptable range.

D. Special Skills Learning

1) Learning of Interleaving Skills: Besides the learning of sequential tasks demonstrated above, LTL progression can also be used to handle interleaving tasks. Consider an interleaving task represented by the following LTL formula:

$$\varphi_{interleaving} = \Diamond\big((\Diamond spam\_grasped \wedge \Diamond\varphi_{push\_jellobox}\big) \\ \wedge\Diamond(bin\_reached \wedge \Diamond spam\_released))$$

where

$$\varphi_{push\_jellobox} = \Diamond(\text{jello}\_pushed \wedge \Diamond shadow\_reached).$$

In English, $\varphi_{\text{interleaving}}$ means "the robot first grasps the spam can spam and pushes the jello box jello at the upper right shadow corner shadow, then takes the grasped spam to *bin*, and releases spam into the *bin*." Note that spam$_$*grasped* and $\varphi_{push\_jellobox}$ can be executed in different orders, and the next subtasks can only be performed when both spam$_$*grasped* and $\varphi_{push\_jellobox}$ are satisfied. Fig. 10(a) and (b) shows the action sketches and snapshots of two different execution sequences of $\varphi_{interleaving}$, respectively, which demonstrate the capability of TRAPs in learning interleaving tasks.

2) Learning of Contact-Rich Skills: To show TRAPs can learn skills with complex dynamics, for example, nonprehensile manipulation or those involving contact-rich interactions, the learning of Peg Insertion skill is considered, which requires the robot to pick up the peg and inserts it into the opening of a wooden block. This process can be described as the LTL formula $\varphi_{peg\_in} = \Diamond(peg\_grasped \wedge \Diamond peg\_inserted)$. Fig. 11(a) shows the action sketches and snapshots for Peg Insertion skill. It can be observed that the robot uses the *grasp* primitive and the *reach* primitive to pick up the peg and aligns it with the hole of the block, respectively. Atomic actions are then used

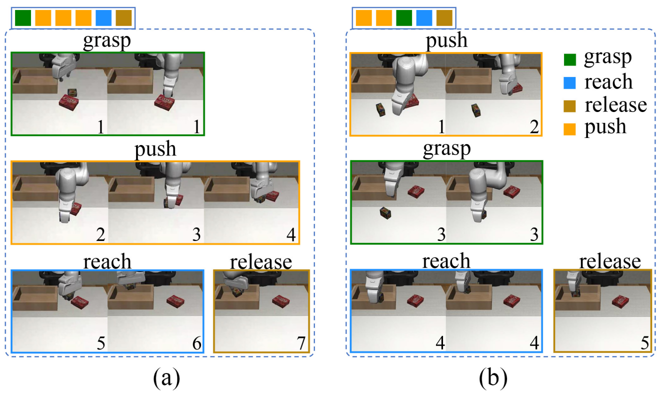


Fig. 10. Action sketches and snapshots for Cleanup skill with interleaving task constraint. (a) and (b) demonstrate two different skill execution sequences with $\varphi_{interleaving}$ constraint, respectively.

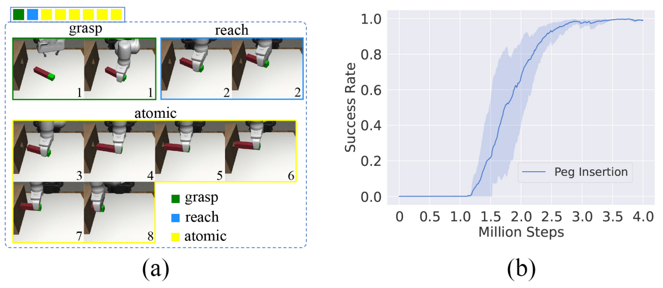


Fig. 11. Learning of Peg Insertion skill. (a) Action sketches and snapshots. (b) Success rate curve in the learning process.

to complete the contact-rich insertion. Fig. 11(b) shows the success rate of learning the Peg Insertion skill.

V. Conclusion

This work develops a TRAPs for robot skill learning. In particular, TRAPs is well suited to long-horizon manipulation skills by augmenting standard RL in two ways. On the one hand, TRAPs uses LTL progression to decompose the training task at an abstract level, informs the robot about their current task progress, and guides them via reward functions. TF-LTL is developed to encode LTL formulas as task latent features to facilitate robot learning with higher expressivity. On the other hand, the PAS, a predefined library of heterogeneous action primitives, further improves the efficiency of robot exploration. Extensive empirical studies demonstrate that TRAPs outperforms most existing methods. Since this work mainly focuses on static tasks and predefined primitive library, ongoing research will consider adaptive task formula inference and dynamic primitive library to enable TRAPs to learn more diverse and challenging missions.

References

[1] O. M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.
[2] G. Xiang and J. Su, "Task-oriented deep reinforcement learning for robotic skill acquisition and control," *IEEE Trans. Cybern.*, vol. 51, no. 2, pp. 1056–1069, Feb. 2021.
[3] G. A. Odesanmi, Q. Wang, and J. Mai, "Skill learning framework for human–robot interaction and manipulation tasks," *Robot. Comput.-Integr. Manuf.*, vol. 79, Feb. 2023, Art. no. 102444.
[4] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Proces. Syst.*, vol. 31, 2018, pp. 1–11.
[5] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," in *Proc. Int. Conf. Machin. Learn*, 2018, pp. 1009–1018.
[6] X. Yang et al., "Hierarchical reinforcement learning with universal policies for multistep robotic manipulation," *IEEE Trans. Neural Netw. Learn. Sys.*, vol. 33, no. 9, pp. 4727–4741, Sep. 2022.
[7] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2018, pp. 452–461.
[8] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith, "LTL2Action: Generalizing LTL instructions for multi-task RL," in *Proc. Int. Conf. Mach. Learn*, 2021, pp. 10497–10508.
[9] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7973–7980, Oct. 2021.
[10] S. Nasiriany, H. Liu, and Y. Zhu, "Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks," in *Proc. Int. Conf. Robot. Autom.*, Philadelphia, PA, USA, 2022, pp. 7477–7484.
[11] M. Guo and M. Bürger, "Geometric task networks: Learning efficient and explainable skill coordination for object manipulation," *IEEE Trans. Robot.*, vol. 38, no. 3, pp. 1723–1734, Jun. 2022.
[12] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *Int. J. Robot. Res.*, vol. 40, nos. 6–7, pp. 866–894, 2021.
[13] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Trans. Neural Netw. Learn. Sys.*, vol. 29, no. 11, pp. 5174–5184, Nov. 2018.
[14] Y. Hu, G. Chen, Z. Li, and A. Knoll, "Robot policy improvement with natural evolution strategies for stable nonlinear dynamical system," *IEEE Trans. Cybern.*, vol. 53, no. 6, pp. 4002–4014, Jun. 2023.
[15] X. Yang, H. Zhang, Z. Wang, H. Yan, and C. Zhang, "Data-based predictive control via multistep policy gradient reinforcement learning," *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 2818–2828, May 2023.
[16] T. Shankar, S. Tulsiani, L. Pinto, and A. Gupta, "Discovering motor programs by recomposing demonstrations," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12.
[17] K. Rana, M. Xu, B. Tidd, M. Milford, and N. Suenderhauf, "Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics," in *Proc. Conf. Robot. Learn*, 2023, pp. 2095–2104.
[18] J. Yamada et al., "Motion planner augmented reinforcement learning for robot manipulation in obstructed environments," in *Proc. Conf. Robot. Learn*, 2021, pp. 589–603.
[19] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *Proc. Int. Conf. Robot. Autom*, 2020, pp. 1–8.
[20] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—A survey," *IEEE Trans. Robot.*, vol. 30, no. 2, pp. 289–309, Apr. 2014.
[21] X. Li, Z. Xu, S. Li, Z. Su, and X. Zhou, "Simultaneous obstacle avoidance and target tracking of multiple wheeled mobile robots with certified safety," *IEEE Trans. Cybern.*, vol. 52, no. 11, pp. 11859–11873, Nov. 2022.
[22] H. Wang, H. He, W. Shang, and Z. Kan, "Temporal logic guided motion primitives for complex manipulation tasks with user preferences," in *Proc. Int. Conf. Robot. Autom*, 2022, pp. 4305–4311.
[23] M. Dalal, D. Pathak, and R. R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 21847–21859.
[24] H. Fu, S. Yu, S. Tiwari, G. Konidaris, and M. Littman, "Meta-learning parameterized skills," 2023, *arXiv:2206.03597v3*.
[25] M. Cai, E. Aasi, C. Belta, and C.-I. Vasile, "Overcoming exploration: Deep reinforcement learning for continuous control in cluttered environments from temporal logic specifications," *IEEE Robot. Autom. Lett.*, vol. 8, no. 4, pp. 2158–2165, Apr. 2023.
[26] M. Cai, S. Xiao, Z. Li, and Z. Kan, "Optimal probabilistic motion planning with potential infeasible LTL constraints," *IEEE Trans. Autom. Control.*, vol. 68, no. 1, pp. 301–316, Jan. 2023.
[27] M. Cai, H. Peng, Z. Li, and Z. Kan, "Learning-based probabilistic LTL motion planning with environment and motion uncertainties," *IEEE Trans. Autom. Control*, vol. 66, no. 5, pp. 2386–2392, May 2021.

[28] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 2, pp. 115–140, May 2019.

[29] C. Innes and S. Ramamoorthy, "Elaborating on learned demonstrations with temporal logic specifications," in *Proc. Robot. Sci. Syst.*, 2020, pp. 1–10.

[30] B. Araki, J. Choi, L. Chin, X. Li, and D. Rus, "Learning policies by learning rules," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 1284–1291, Apr. 2022.

[31] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[32] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods Syst. Design*, vol. 19, no. 3, pp. 291–314, 2001.

[33] B. Lacerda, D. Parker, and N. Hawes, "Optimal policy generation for partially satisfiable co-safe LTL specifications," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1–7.

[34] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1–7.

[35] Y.-L. Kuo, B. Katz, and A. Barbu, "Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas," in *Proc. Int. Conf. Intell. Robot. Syst*, 2020, pp. 5604–5610.

[36] B. G. León, M. Shanahan, and F. Belardinelli, "Systematic generalisation through task temporal logic and deep reinforcement learning," 2020, *arXiv:2006.08767*.

[37] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artif. Intell.*, vol. 116, nos. 1–2, pp. 123–191, 2000.

[38] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Proces. Syst.*, vol. 30, 2017, pp. 1–15.

[39] H. Zhang, H. Wang, and Z. Kan, "Exploiting transformer in reinforcement learning for interpretable temporal logic motion planning," 2022, *arXiv:2209.13220*.

[40] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor–critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[41] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Proc. Conf. Robot. Learn*, 2021, pp. 188–204.

[42] Y. Zhu et al., "Robosuite: A modular simulation framework and benchmark for robot learning," 2020, *arXiv:2009.12293*.

[43] S. Zhang and S. Whiteson, "DAC: The double actor–critic architecture for learning options," in *Proc. Adv. Neural Inf. Proces. Syst.*, vol. 32, 2019, pp. 1–15.

[44] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta, "Efficient bimanual manipulation using learned task schemas," in *Proc. Int. Conf. Robot. Autom*, 2020, pp. 1149–1155.

[45] Y. Lee, J. Yang, and J. J. Lim, "Learning to coordinate manipulation skills via skill behavior diversification," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–15.

[46] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Phys. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

Hao Wang received the B.S. degree in mechanical engineering from the China University of Mining and Technology, Xuzhou, Jiangsu, China, in 2017. He is currently pursuing the Ph.D. degree in automation with the University of Science and Technology of China, Hefei, Anhui, China.

His current research interests include motion planning in robotics, manipulation skill learning, and deep reinforcement learning.

Hao Zhang received the B.S. degree in mechanical engineering and automation from the Hefei University of Technology, Hefei, Anhui, China, in 2020. He is currently pursuing the Ph.D. degree in automation with the University of Science and Technology of China, Hefei.

His current research interests include formal methods in robotics, reinforcement learning, and dexterous manipulation.

Lin Li received the B.S. degree in mechanical engineering and automation from the Hefei University of Technology, Hefei, China, in 2021. She is currently pursuing the M.S. degree in control engineering with the University of Science and Technology of China, Hefei.

Her current research interests include heterogeneous multirobot system and motion planning.

Zhen Kan (Senior Member, IEEE) received the Ph.D. degree from the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, USA, in 2011.

He is currently a Professor with the Department of Automation, University of Science and Technology of China, Hefei, China. He was a Postdoctoral Research Fellow with Air Force Research Laboratory, Eglin AFB, FL, USA, and the University of Florida REEF, Shalimar, FL, USA, from 2012 to 2016, and an Assistant Professor with the Department of Mechanical Engineering, University of Iowa, Iowa City, IA, USA, from 2016 to 2019. His research interests include networked control systems, nonlinear control, formal methods, and robotics.

Prof. Kan currently serves on the program committees of several internationally recognized scientific and engineering conferences and is an Associate Editor of IEEE TRANSACTIONS ON AUTOMATIC CONTROL.

Yongduan Song (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from Tennessee Technological University, Cookeville, TN, USA, in 1992.

He is currently the Dean of the Research Institute of Artificial Intelligence, Chongqing University, Chongqing, China.

Prof. Song is the Editor-in-Chief of IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS and the Founding Editor-in-Chief of the *International Journal of Automation and Intelligence*. He was one of the six Langley Distinguished Professors at the National Institute of Aerospace, USA and a Register Professional Engineer (USA). He is a Fellow of AAIA, the International Eurasian Academy of Sciences, and the Chinese Automation Association.