LTL-Transfer: Skill Transfer for Temporal Task Specification

Jason Xinyu Liu*[1], Ankit Shah*[1], Eric Rosen[1], Mingxi Jia[1], George Konidaris[1] and Stefanie Tellex[1]

***Abstract*— Deploying robots in real-world environments, such as households and manufacturing lines, requires generalization across novel task specifications without violating safety constraints. Linear temporal logic (LTL) is a widely used task specification language with a compositional grammar that naturally induces commonalities among tasks while preserving safety guarantees. However, most prior work on reinforcement learning with LTL specifications treats every new task independently, thus requiring large amounts of training data to generalize. We propose LTL-Transfer, a zero-shot transfer algorithm that composes task-agnostic skills learned during training to safely satisfy a wide variety of novel LTL task specifications. Experiments in Minecraft-inspired domains show that after training on only 50 tasks, LTL-Transfer can solve over 90% of 100 challenging unseen tasks and 100% of 300 commonly used novel tasks without violating any safety constraints. We deployed LTL-Transfer at the task-planning level of a quadruped mobile manipulator to demonstrate its zero-shot transfer ability for fetch-and-deliver and navigation tasks.**

I. INTRODUCTION

Deploying robots in the real world requires generalization across many novel tasks while preserving safety. For example, an industrial robot that fetches the same components in different orders based on the assembled product should only learn to fetch each part once. These tasks typically share constituents like objects and trajectory segments, which creates an opportunity to reuse knowledge [1].

Linear temporal logic (LTL) [2] is an effective means of specifying objectives, including safety constraints, for reinforcement learning (RL) agents [3], [4], [5]. Its compositional grammar reflects the compositional nature of most tasks. However, most prior approaches to RL for LTL tasks learn to solve every new task from scratch. We propose a zero-shot transfer algorithm, LTL-Transfer, that exploits the compositionality of LTL task specification to safely solve novel tasks without additional training by composing skills learned in prior tasks. Transferring skills is more data-efficient than learning from scratch and more computationally efficient than planning.

We show in a simulated Minecraft-inspired domain that LTL-Transfer can solve over 90% of 100 challenging unseen tasks and 100% of 300 common and less complex novel tasks after training on only 50 tasks and never violates a safety constraint. We deploy LTL-Transfer at the task-planning level of a quadruped mobile manipulator to solve fetch-and-deliver and navigation tasks in zero-shot. Our key insight is efficiently reusing learned skills by leveraging similarities between the novel and training tasks. Code, datasets, supplementary materials, and robot demonstration videos are at `https://jasonxyliu.github.io/LTL-Transfer`

*Equal contribution. [1] Brown University.

(a) go to shelf to fetch book (b) deliver juice to desk

Fig. 1: The robot is executing four task-agnostic skills sequentially to solve a novel task $\mathbf{F}(book \wedge \mathbf{F}(desk_a \wedge \mathbf{F}(juice \wedge \mathbf{F}desk_a)))$, i.e., fetch and deliver a book then a juice bottle to the user. Two of the four skills are shown.

II. PRELIMINARIES

Linear Temporal Logic (LTL) for Task Specification: LTL is a widely used alternative to numerical rewards for task specification. An LTL formula φ is a Boolean function that determines whether a given trajectory satisfies the objective expressed by the formula. Littman et al. [3] showed that LTL can express non-Markovian, temporal tasks that numerical rewards cannot, and it has become a target language for acquiring task specification in many settings, including from natural language [6], [7] and learning from demonstration [8]. Formally, an LTL formula is interpreted over traces of Boolean propositions over discrete time and is defined through the following recursive syntax:

$$\varphi := \alpha \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \ \mathbf{U} \ \varphi_2. \tag{1}$$

$\alpha \in \boldsymbol{\alpha}$ is a proposition that maps a state to a Boolean value; φ, φ_1, and φ_2 are valid LTL formulas. The operator $\mathbf{X}$ (next) is used to define a property $\mathbf{X}\varphi$ that holds if φ holds at the next time step. The formula $\varphi_1 \ \mathbf{U} \ \varphi_2$ with the binary operator $\mathbf{U}$ (until) specifies that φ_1 must hold until φ_2 first holds at a future time. The operators $\neg$ (not) and $\vee$ (or) are identical to propositional logic operators. We also utilize the following abbreviated operators: $\wedge$ (and), $\mathbf{F}$ (finally or eventually), and $\mathbf{G}$ (globally or always). $\mathbf{F}\varphi$ specifies that the formula φ must hold at least once in the future, and $\mathbf{G}\varphi$ specifies that φ must always hold in the future. Consider the Minecraft map depicted in Figure 2. The task of collecting $wood$ and axe in an arbitrary order is specified by the LTL formula $\mathbf{F}axe \wedge \mathbf{F}wood$. The formula $\mathbf{F}(axe \wedge \mathbf{F}wood)$ specifies collecting $wood$ at least once after collecting axe. $\mathbf{F}wood \wedge \neg wood \ \mathbf{U} \ axe$ specifies the task of collecting $wood$ only after axe has been collected.

Every LTL formula can be represented as a Büchi automaton [9], [10] interpreted over an infinite trace of truth values of the propositions in the formula, thus providing an automated

translation of an LTL specification to a transition-based representation. We consider task specification in the co-safe fragment of LTL [11], [12] where formulas can be verified by a finite trace, thus making it ideal for episodic tasks. Camacho et al. [5] showed that every co-safe LTL formula φ can be translated to an equivalent reward machine (RM) [13], [14], $\mathcal{M}_\varphi = \langle \mathcal{Q}_\varphi, q_{0,\varphi}, \mathcal{Q}_{term,\varphi}, \varphi, T_\varphi, R_\varphi \rangle$; where $\mathcal{Q}_\varphi$ is the finite set of states, $q_{0,\varphi}$ is the initial state, $\mathcal{Q}_{term,\varphi}$ is the set of terminal states; $T_\varphi : \mathcal{Q}_\varphi \times 2^{|\boldsymbol{\alpha}|} \to \mathcal{Q}_\varphi$ is the deterministic transition function; and $R_\varphi : \mathcal{Q}_\varphi \to \mathbb{R}$ represents the reward accumulated by entering a given state. Figure 2d shows the reward machine graph representing the LTL formula $\mathbf{F}wood \wedge \neg wood\ \mathbf{U} axe$. Nodes encode progressed RM states, 0 for an accepting state and 3 for a failure state. Boolean formulas label edges. Truth assignments of propositions $\boldsymbol{\alpha}$ that satisfy edges induce transitions in the RM. Our proposed algorithm, LTL-Transfer, for transferring learned policies to solve novel LTL specifications, is compatible with all algorithms that generate policies by solving a product MDP of the reward machine $\mathcal{M}_\varphi$ and the task environment.

Options Framework: Sutton and Barto [15] introduced a framework, termed options or skills, for incorporating temporally extended actions into reinforcement learning. An option $o = \langle \mathcal{I}, \beta, \pi \rangle$ is defined by the initiation set $\mathcal{I}$, a set of states where the option policy can be executed; the termination condition β, which determines when the execution ends; and the option policy π. Our proposed algorithm, LTL-Transfer, compiles policies learned during training into task-agnostic options that are transferred to solve novel tasks.

III. Related work

Most approaches extending reinforcement learning (RL) to temporal tasks first generate a product MDP of the state space and the automaton equivalent of the LTL task specification [3], [14], [4], [5], [13]. Notably, Jothimurugan et al. [16] proposed interleaving graph-based planning on the automaton with RL to bias exploration towards trajectories that satisfy the LTL specification. Although these approaches exploited the compositional structure of LTL to accelerate learning, they did not exploit the compositionality to transfer to novel tasks. Thus, the policy to satisfy a novel LTL specification must be learned from scratch.

A common approach towards generalization across temporal tasks has been to learn independent policies for subtasks [17], [18], [19], [20]. Given a new task, these methods then sequentially compose the learned policies in an admissible order. Consider the Minecraft-inspired grid world depicted in Figure 2a containing $wood$ and axe objects. The subtask-based methods learn policies to solve subtasks $\mathbf{F}wood$ and $\mathbf{F}axe$ involving reaching each object. When tasked with the specification $\varphi_{test} = \mathbf{F}wood \wedge (\neg wood\ \mathbf{U}\ axe)$, i.e., collect $wood$, but do not collect $wood$ until axe is collected, the subtask-based methods would violate the ordering constraint by reaching axe through the grid cells containing $wood$. These approaches rely on additional fine-tuning to solve the target task correctly. We propose a general framework for transferring learned policies to novel tasks in zero-shot without violating any safety constraints.

Kuo et al. [21] proposed learning subnetworks for propositions and operators, then creating the final policy network through composition. Vaezipoor et al. [22] proposed learning a latent embedding over LTL formulas using a graph neural network to solve novel LTL tasks. In contrast, our method uses formal methods to identify learned policies best suited for transfer, thus requiring orders of magnitude fewer training tasks to achieve comparable results. Furthermore, neither method can guarantee the preservation of safety constraints like our approach. Xu et al. [23] considered transfer learning between pairs of source and target tasks, while our approach trains on a collection of tasks and transfers to a set of novel tasks. Nangue Tasse et al. [24] attempted zero-shot composition through logical and temporal compositions of policies, but their approach assumes that a given task must be satisfiable. By leveraging the structure of the task automaton, our proposed algorithm aborts execution immediately after it identifies the task as unsolvable given the learned skills.

A qualitatively distinct approach to zero-shot generalize to novel tasks is model-based RL that plans with an estimated transition model [25], [26]. Compared to model-based methods, our approach transfers learned policies in zero-shot thus requires significantly less computation at test time.

Our approach draws inspiration from prior works on learning portable skills in Markov domains [27], [28], [29], [30]. These approaches learn task-agnostic representations of preconditions, constraints, and effects of an option [15]. We learn portable skills to satisfy novel LTL task specifications.

IV. Problem definition

We represent the environment as an MDP without the reward function $\mathcal{M}_\mathcal{S} = \langle \mathcal{S}, \mathcal{A}, T_\mathcal{S} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and $T_\mathcal{S} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ represents the transition dynamics of the environment which we assume to be hidden from the RL algorithm. Further, a set $\boldsymbol{\alpha}$ of Boolean propositions represents the facts about the environment and forms the compositional building blocks for specifying tasks. We assume a labeling function $L : \mathcal{S} \to 2^{|\boldsymbol{\alpha}|}$ that maps the state to the Boolean propositions is given. A task in the environment $\mathcal{M}_\mathcal{S}$ is specified by a linear temporal logic (LTL) formula φ, and it is translated to a reward machine $\mathcal{M}_\varphi = \langle \mathcal{Q}_\varphi, q_{0,\varphi}, \mathcal{Q}_{term,\varphi}, \varphi, T_\varphi, R_\varphi \rangle$ detailed in Section II.

Given a set of training tasks $\Phi_{train} = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$, specified by LTL, and policies for a set of options $\mathcal{O}_q$ learned from these training tasks, a zero-shot transfer algorithm needs to solve a novel LTL task $\varphi_{test} \notin \Phi_{train}$ in the same environment without additional training.

V. LTL-Transfer with transition-centric options

A. Algorithm Overview

Consider the environment map depicted in Figure 2b. Assume an RL algorithm has learned option policies to collect axe and $wood$ individually from the training tasks $\mathbf{F}axe$ and $\mathbf{F}wood$, respectively. Now given the test task $\varphi_1 = \mathbf{F}(axe \wedge \mathbf{F}\ wood)$, i.e., first collect axe, then $wood$, a

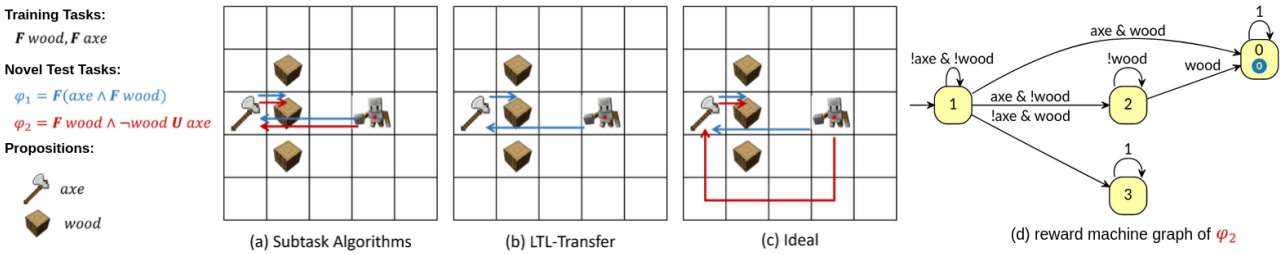


Fig. 2: An example of tasks, the environment, and trajectories. The robot learned to solve the two training LTL tasks and is expected to solve two novel tasks φ_1 and φ_2. Figure 2a depicts the trajectories output by a subtask-based algorithm (blue for φ_1, red for φ_2). Figure 2b depicts the trajectories produced by our proposed algorithm LTL-Transfer. Note that LTL-Transfer does not start execution for the task φ_2 as the two learned policies do not guarantee the preservation of the ordering constraint. Figure 2c depicts the optimal trajectories for the novel tasks φ_1 and φ_2. Figure 2d is a graph representation of the reward machine (RM) for the task φ_2. Nodes represent RM states. Edges represent Boolean formulas.

transfer algorithm should identify that sequentially composing the policies for $\mathbf{F}axe$ and $\mathbf{F}wood$ solves the new task φ_1 (as depicted in blue). Then consider a different test task $\varphi_2 = \mathbf{F}wood \ \wedge \ \neg wood \ \mathbf{U}axe$, i.e., first collect axe, then $wood$, but avoid $wood$ until axe is collected. The transfer algorithm must realize that the policy that satisfies $\mathbf{F}axe$ does not guarantee avoiding $wood$ while going to axe. Therefore, it must not start execution using only these two learned policies to avoid accidentally violating the ordering constraint.

We developed a zero-shot transfer algorithm, LTL-Transfer, that composes learned policies to solve novel LTL tasks while enforcing ordering constraints. It operates in two stages.

1) First, LTL-Transfer accepts the set of training tasks Φ_{train} and the policies learned from the training tasks and compiles a set of task-agnostic, portable options $\mathcal{O}_e$.
2) Next, it identifies and executes a sequence of options from the set $\mathcal{O}_e$ to solve a novel task φ_{test}.

We can use a class of RL algorithms that operate on a product MDP composed of the environment $\mathcal{M}_\mathcal{S}$ and the reward machine $\mathcal{M}_\varphi$ to learn option policies from the training LTL tasks [3], [14], [4], [5], [13]. We chose LPOPL [4] because it explicitly allows for sharing policies across LTL task specifications that share progression states. Given a set of LTL tasks, LPOPL first translates each task to a reward machine (RM) and decomposes tasks to subtasks, each of which corresponds to a state in the RM, then learns an action-value function, represented by a DQN [31], for each subtask. Please see supplementary materials for implementation details of LPOPL. The learned policy is Markov with respect to the environment states $\mathcal{S}$ for a given RM state, i.e., the policy to be executed in the RM state $q \in \mathcal{Q}_\varphi$ is $\pi^\varphi_q : \mathcal{S} \to \mathcal{A}$.

Executing the state-centric option $o^\varphi_q \in \mathcal{O}_q$ with the policy π^φ_q from the reward machine state q triggers a transition in the RM on a path to an acceptance state. There can be multiple outgoing transitions from an RM state, so the target RM transition of an option o^φ_q is conditioned on the environment state $s \in \mathcal{S}$ where the execution was initiated. We propose compiling each state-centric option, o^φ_q, into multiple transition-centric options by partitioning the initiation set of the state-centric option based on the estimated success probability of its policy satisfying the target RM transition from the starting environment state. Each resulting transition-centric option will maintain the satisfaction of self-transition edge $e^\varphi_{q,q}$ from the starting RM state q until it triggers the target RM transition $e^\varphi_{q,q'}$. Our insight is that each transition-centric option triggers a transition in the reward machine on a path to an acceptance state, and these RM transitions may be shared across different tasks. Thus, the transition-centric options $\mathcal{O}_e$ are portable across different tasks. We describe the compilation algorithm in Section V-B.

Given a novel LTL task specification $\varphi_{test} \notin \Phi_{train}$, our transfer algorithm first constructs a reward machine representing the task, $\mathcal{M}_{\varphi_{test}}$, then identifies a path through the reward machine that can be traversed by sequentially executing transition-centric options from the set $\mathcal{O}_e$. Our transfer algorithm is sound, and it terminates. We describe the details of the transfer algorithm in Section V-C.

The key advantage of our approach is that the option compilation can be done offline for any environment. We can then transfer the options to solve novel tasks at execution time in zero-shot. Thus, learning to solve a limited number of LTL tasks can help solve a wide gamut of unseen tasks.

B. Compilation of Transition-Centric Options

The policy learned to satisfy an LTL task specification φ identifies the current reward machine state $q \in \mathcal{Q}_\varphi$ and executes a Markov policy π^φ_q until the state of the reward machine progresses. We represent this policy as a state-centric option, $o^\varphi_q = \langle \mathcal{S}, \beta_{e^\varphi_{q,q}}, \pi^\varphi_q \rangle$, where the initiation set is the entire state space $\mathcal{S}$ of the environment; the option terminates when the truth assignment of the propositions $\boldsymbol{\alpha}$ violates the self-transition $e^\varphi_{q,q}$ from the RM state q. The termination condition $\beta_{e^\varphi_{q,q}}$ is formally defined as follows,

$$\beta_{e^\varphi_{q,q}} = \begin{cases} 1, & \text{if } L(s) \not\models e^\varphi_{q,q} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

A transition-centric option $o_{e^\varphi_{q,q}, e^\varphi_{q,q'}}$ executes a Markov policy that ensures that the truth assignment of the propositions $\boldsymbol{\alpha}$ satisfies the self-transition edge $e^\varphi_{q,q}$ at all time until the policy yields a truth assignment that satisfies the target outgoing edge $e^\varphi_{q,q'}$. We define a transition-centric option as follows,

$$o_{e^\varphi_{q,q}, e^\varphi_{q,q'}} = \langle \mathcal{S}, \beta_{e^\varphi_{q,q}}, \pi^\varphi_q, e^\varphi_{q,q}, e^\varphi_{q,q'}, f_{e^\varphi_{q,q'}} \rangle. \tag{3}$$

Algorithm 1 Compile State-Centric Options to Transition-Centric Options

1: **function** COMPILE($\mathcal{M}_\mathcal{S}, \Phi_{train}, \mathcal{O}_q$)
2: $\quad\mathcal{O}_e \leftarrow \emptyset$
3: $\quad$**for** $\varphi \in \Phi_{train}$ **do**
4: $\quad\quad\mathcal{M}_\varphi \leftarrow$ GENERATERM(φ)
5: $\quad\quad\mathcal{O}_q^\varphi \leftarrow \left\{o_q^{\varphi'} \in \mathcal{O}_q : \varphi' = \varphi\right\}$
6: $\quad\quad$**for** $o_q^\varphi = \langle \mathcal{S}, \beta_{e_{q,q}^\varphi}^\varphi, \pi_q^\varphi \rangle \in \mathcal{O}_q^\varphi$ **do**
7: $\quad\quad\quad\mathcal{Q}_{out} = \{q' : q' \text{ is an out-neighbor of } q\}$
8: $\quad\quad\quad\forall q' \in \mathcal{Q}_{out} : \mathcal{E} \leftarrow \left\{(e_{q,q}^\varphi, e_{q,q'}^\varphi) : e_{q,q}^\varphi \text{ is the self edge}\right\}$
9: $\quad\quad\quad$**for** $s \in \mathcal{S}$ **do**
10: $\quad\quad\quad\quad$Generate N_r rollouts from s using π_q^φ
11: $\quad\quad\quad\quad$Record edge transition frequencies $n_s(e_{q,q'}^\varphi)\ \forall\ (e_{q,q}^\varphi, e_{q,q'}^\varphi) \in \mathcal{E}$
12: $\quad\quad\quad\quad\forall\ q' \in \mathcal{Q}_{out} : f_{e_{q,q'}^\varphi}(s) \leftarrow \frac{n_s(e_{q,q'}^\varphi)}{N_r}$
13: $\quad\quad\quad\mathcal{O}_{q,e}^\varphi \leftarrow \left\{o_{e_{q,q}^\varphi, e_{q,q'}^\varphi} = \langle \mathcal{S}, \beta_{e_{q,q}^\varphi}, \pi_q^\varphi, e_{q,q}^\varphi, e_{q,q'}^\varphi, fe_{q,q'}^\varphi \rangle\right\}$
14: $\quad\quad\quad\mathcal{O}_e \leftarrow \mathcal{O}_e \cup \mathcal{O}_{q,e}^\varphi$
15: $\quad$**return** $\mathcal{O}_e$

Algorithm 2 Zero-shot transfer to test task φ_{test}

1: **function** TRANSFER($\mathcal{M}_\mathcal{S}$, φ_{test}, $\mathcal{O}_e$)
2: $\quad\mathcal{M}_{\varphi_{test}} \leftarrow$ GENERATE_RM(φ_{test})
3: $\quad\mathcal{M}_{\varphi_{test}} \leftarrow$ PRUNE($\mathcal{M}_{\varphi_{test}}$)
4: $\quad s \leftarrow$ INITIALIZE($\mathcal{M}_\mathcal{S}$)
5: $\quad q \leftarrow q_{0,\varphi_{test}}$
6: $\quad$**while** $q \neq q^\top$ **do**
7: $\quad\quad P_{\text{cache}} \leftarrow \left\{p_i :\ p_i = [e_0, \ldots, e_{n_i}] \text{ path from } q \text{ to } q^\top \text{ in } \mathcal{M}_{\varphi_{test}}\right\}$
8: $\quad\quad\mathcal{O}_{p[0]} = \left\{o_{e_1,e_2} \in \mathcal{O}_e : \text{MATCHEDGE}((e_1, e_2), (e_{q,q}^{\varphi_{test}}, p[0]))\right\}$ $\forall p \in P_{\text{cache}}$
9: $\quad\quad\mathcal{O}_{[0]} = \bigcup_p \mathcal{O}_{p[0]}$
10: $\quad\quad\langle s', q' \rangle \leftarrow \langle s, q \rangle$
11: $\quad\quad$**while** $\mathcal{O}_{[0]} \neq \emptyset$ and $q' = q$ **do**
12: $\quad\quad\quad o_{e_1,e_2}^* \leftarrow \arg\max_{o_{e_1,e_2} \in \mathcal{O}_{[0]}} f_{e_2}(s)$
13: $\quad\quad\quad\langle s', q' \rangle \leftarrow$ EXECUTE(π^*)
14: $\quad\quad\quad$**if** $q' = q$ **then**
15: $\quad\quad\quad\quad\mathcal{O}_{[0]} \leftarrow \mathcal{O}_{[0]} \setminus o_{e_1,e_2}^*$
16: $\quad\quad$**if** $q' = q$ **then**
17: $\quad\quad\quad$**return** Failure
18: $\quad\quad$**else**
19: $\quad\quad\quad\langle s, q \rangle \leftarrow \langle s', q' \rangle$
20: $\quad$**return** Success

The initiation set is the entire state space $\mathcal{S}$ of the environment; the termination condition $\beta_{e_{q,q}^\varphi}$ is defined by the violation of the self-transition edge $e_{q,q}^\varphi$; the option's policy is Markov $\pi_q^\varphi : \mathcal{S} \to \mathcal{A}$; $e_{q,q}^\varphi$ and $e_{q,q'}^\varphi$ represent the self-transition and the target outgoing edge, respectively; and $f_{e_{q,q'}^\varphi} : \mathcal{S} \to [0, 1]$ represents the success probability of the policy π_q^φ satisfying the target edge $e_{q,q'}^\varphi$ starting from the environment state $s \in \mathcal{S}$.

Algorithm 1 describes our approach to compiling each state-centric option o_q^φ to a set of transition-options $\left\{o_{e_{q,q}^\varphi, e_{q,q'}^\varphi} : q \in \mathcal{Q}_\varphi, q' \text{is out-neighbor of } q\right\}$. Executing the policy π_q^φ of the state-centric option may satisfy different outgoing edges $e_{q,q'}^\varphi$ of the reward machine state q depending on what environment state $s \in \mathcal{S}$ the execution was initiated. Thus, the success probability $f_{e_{q,q'}^\varphi}$ acts as a soft segmenter of the state space $\mathcal{S}$; it can be estimated by policy rollouts from all environment states in discrete domains or using sampling-based methods [29], [30] in continuous domains.

C. Transferring to Novel LTL Task Specification

Algorithm 2 describes the zero-shot transfer algorithm that composes transition-centric options from the set $\mathcal{O}_e$ to solve a novel test task φ_{test} in the environment $\mathcal{M}_\mathcal{S}$. Line 2 generates the reward machine (RM) graph for the test task. Line 3 examines each edge $e_{q,q'}^{\varphi_{test}}$ of the RM, identifies the transition-centric options that satisfy that edge transition, and prunes an edge if no such option is found. Line 7 identifies and caches all paths from the current state q to the accepting state $q^\top$ of the reward machine. Lines 8 and 9 identify a set of all eligible options that can potentially achieve an RM transition from the current state to a progressed state on a path to the accepting state.

Lines 12 and 13 then execute the option with the highest success probability of satisfying the target edge transition, estimated by f. If the option fails to progress to another RM state, we delete it from the set (Line 15) and execute the next option. If the set of eligible options is empty at any point without reaching the accepting RM state $q^\top$, Line 17 exits with a failure. A successful transfer occurs when the RM progresses to the accepting state $q^\top$.

D. Matching Options to Reward Machine Transitions

The edge-matching conditions determine whether we can safely apply a transition-centric option to transition along an edge of the reward machine (RM). We used the edge-matching conditions to prune the RM graph to retain only the edges matched with available options (Algorithm 2 Line 3) and identify eligible options from a given RM state (Algorithm 2 Line 8). Given a test task φ_{test}, when the current RM state is q, its self-transition edge is $e_{q,q}^{\varphi_{test}}$ and the target outgoing edge is $e_{q,q'}^{\varphi_{test}}$, to determine if a transition-centric option o_{e_1,e_2} matches the target transition in the RM, we propose two edge-matching conditions, *Constrained* and *Relaxed*. Both ensure the task execution does not fail due to an unsafe transition.

Constrained Edge-Matching Condition requires that every truth assignment satisfying the self-transition edge e_1 of the option also satisfies the self-transition edge $e_{q,q}^{\varphi_{test}}$ of the reward machine $\mathcal{M}_{\varphi_{test}}$. Similarly, every truth assignment satisfying the outgoing edge e_2 of the option must satisfy the target transition $e_{q,q'}^{\varphi_{test}}$ of the test task's RM. This strict requirement reduces the applicability of the learned options but ensures that the target edge is always achieved. The *Constrained* edge-matching condition is satisfied if the following Boolean expression holds:

$$\neg sat(e_1 \wedge \neg e_{q,q}^{\varphi_{test}}) \ \wedge\ \neg sat(e_2 \wedge \neg e_{q,q'}^{\varphi_{test}}). \tag{4}$$

Let $sat(g)$ be a Boolean function that is true if and only if a truth assignment exists to satisfy the Boolean formula g.

Relaxed Edge-Matching Condition requires the self edges e_1 and $e_{q,q}^{\varphi_{test}}$ share satisfying truth assignments, so must the outgoing edges e_2 and $e_{q,q'}^{\varphi_{test}}$. However, it allows the option to have valid truth assignments that may not satisfy the target outgoing edge, yet none of the truth assignments should trigger a transition to an unrecoverable failure state $q^\perp$. We identify $q^\perp$ as a sink state of the RM graph without outgoing edges. Further, all truth assignments that terminate the option must not satisfy the self-transition edge of the test task's reward machine. The *Relaxed* edge-matching condition can

retrieve more eligible options. It is satisfied if the following Boolean expression holds:

$$sat(e_1 \wedge e_{q,q}^{\varphi_{test}}) \ \wedge \ sat(e_2 \wedge e_{q,q'}^{\varphi_{test}}) \wedge \\ \neg sat(e_1 \wedge e^{\perp}) \ \wedge \ \neg sat(e_2 \wedge e^{\perp}) \ \wedge \ \neg sat(e_2 \wedge e_{q,q}^{\varphi_{test}}). \tag{5}$$

E. Optimization

We parallelized the two key computational bottlenecks of LTL-Transfer, i.e., the estimation of the success probability $f_{e^{\varphi}_{q,q'}}$ and the evaluation of the *Relaxed* edge-matching condition since there is no shared memory requirements. We implemented the *Relaxed* edge-matching condition using a propositional model counting algorithm [32] from SymPy [33], and the *Constrained* edge-matching condition using string comparison to circumvent the model counting problem and found a significant speed-up.

VI. Experiments

The aim of our evaluation is to test the hypothesis that LTL-Transfer can efficiently transfer learned skills to solve novel temporal tasks while preserving safety guarantees. We tested our hypothesis in simulation and on a physical robot. The simulation domain allows us to scale skill transfer across a wide variety of tasks, while the real robot demonstrates the practicality of our approach for real-world mobile manipulation tasks. We first defined five types of specifications of varying complexity for training and testing, then conducted experiments to evaluate the following hypotheses,

1) **H1:** LTL-Transfer exceeds the baselines' success rates of solving novel tasks.
2) **H2:** *Relaxed* edge-matching condition leads to higher success rates than the *Constrained* condition.
3) **H3:** Transferring learned policies to certain specification types (introduced in Section VI-B) leads to higher success rates.
4) **H4:** LTL-Transfer is robust to highly uncertain transition dynamics.

A. Task Environment

We tested LTL-Transfer in a Minecraft-inspired grid-world domain commonly seen in compositional reinforcement learning (RL) and integration of temporal logics with RL [20], [4], [16], [19]. This domain is particularly well suited for testing transfer learning with temporal tasks as policies to solve individual tasks can be learned rapidly with few computational resources. Thus, we can run comprehensive evaluations of skill transfer across a full factorial of training and test task types that cover a wide variety of LTL templates. We conducted experiments on four maps, similar to that depicted in Figure 2, with a dimension of 19×19. Each location in the map is either vacant or occupied by one of nine object types. Multiple instances of an object type may occur across the map. After the agent enters a grid cell occupied by an object, the proposition representing that object type becomes true. Actions are moving in four cardinal directions; an invalid action results in no movement. We tested zero-shot transfer in both deterministic and stochastic domains.

B. Types of Task Specifications

For a comprehensive evaluation of transferring learned policies across various LTL specifications, we considered the following three types of ordering constraints, proposed by Shah et al. [8], which constitute five specification types. Each constraint is defined on a pair of propositions a and b. Without loss of generality, we assume that a precedes b.

1) **Hard** ordering constraints occur when b must never be true before a. This property is expressed through the LTL formula $\neg b \ \mathbf{U} \ a$.
2) **Soft** ordering constraints allow b to occur before a as long as b happens at least once after a becomes true for the first time. Soft orders are expressed through the LTL formula $\mathbf{F}(a \ \wedge \ \mathbf{F}b)$.
3) **Strictly Soft** ordering constraints are similar to soft orders except that b must be true strictly after a first holds. Thus, a and b holding simultaneously would not satisfy a strictly soft order. Strictly soft orders are expressed through the LTL formula $\mathbf{F}(a \ \wedge \ \mathbf{XF}b)$.

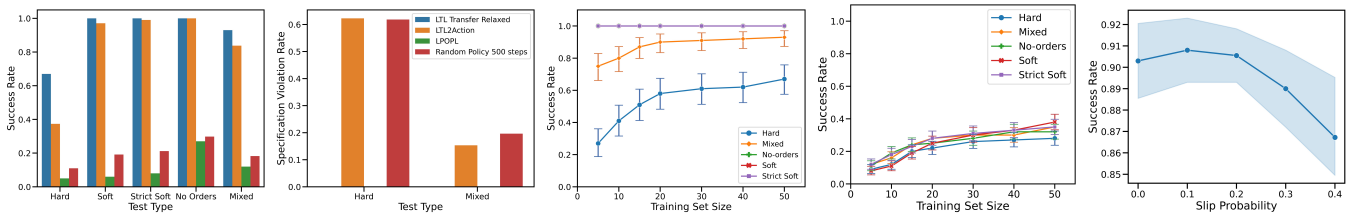
We sampled five training sets, with 50 formulas in each, that represent five different specification types, *Hard*, *Soft*, *Strictly-Soft*, *No-Orders*, and *Mixed*, and a test set of 100 formulas for each type. This imitates the real-world scenario where users do not know the test task beforehand, so the robot must be trained on a limited set of training tasks then transfer to a wide variety of novel tasks. When constructing a test set, we excluded tasks already in the training set. All specifications in the *Hard*, *Soft*, and *Strictly-Soft* sets were expressed using the respective templates described above. *No-Orders* sets have no ordering constraints, e.g., $\mathbf{F}a \ \wedge \ \mathbf{F}b \ \wedge \ \mathbf{F}c$. In the *Mixed* set, each binary precedence constraint was expressed as one of the three ordering constraints described above. A given task in the environment involves visiting a specified set of object types in an admissible order determined by ordering constraints. Please see supplementary materials for example formulas from each specification type.

C. Results and Discussion

Comparison with Baselines: We compare the performance of LTL-Transfer to three baselines, i.e., LTL2Action [22], LPOPL [4], and a random policy.

LTL2Action proposed by Vaezipoor et al. [22] embeds LTL specifications using a graph neural network and sequentially selects the next proposition to satisfy. This pre-trained embedding is appended to the state features to yield a goal-conditioned task policy, termed LTL Bootcamp, which serves as the upper bound of LTL2Action's performance on novel tasks. We trained the LTL Bootcamp on the same training sets as LTL-Transfer and compared their performance.

LPOPL, detailed in Section V-A, serves as the lower bound that any transfer algorithm must surpass due to its limited transfer ability to solve novel tasks. While LPOPL was not explicitly designed for zero-shot transfer, it can satisfy task specifications that are a progression of a training LTL formula because of its use of LTL progression and multi-task learning.



(a) Success Rate (b) Violation Rate (c) *Relaxed* Match (d) *Constrained* Match (e) Stochastic Domain

Fig. 3: Figure 3a shows the success rates of four methods on five test sets after training on the *Mixed* training set. Figure 3b depicts their specification violation rates (with a shared legend). Figure 3d and 3c show the success rates of LTL-Transfer on five test sets after training on *Mixed* sets of various sizes using *Relaxed* and *Constrained* edge-matching condition, respectively. The error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution. Figure 3e shows the success rate as the slip probability increases averaged over four maps.

Figure 3a depicts the success rate of all the methods on 100 novel tasks in each test set described in Section VI-B after training on 50 tasks of the *Mixed* type. LPOPL performed worse than the random policy as it did not attempt to satisfy any specification that did not exist in its progression set. Both LTL-Transfer and LTL2Action have near-perfect success rates on *Soft*, *Strictly-Soft*, and *No-Orders* test set as these specifications do not have irrecoverable failure state. Specifications in *Hard* and *Mixed* test sets have failure states. Thus, we observe a lower success rate across all methods. However, LTL-Transfer demonstrates the best transfer success rate in the difficult test sets. Crucially, Figure 3b shows that LTL-Transfer never violated any specification, while LTL2Action's specification violation rate is similar to that of the random policy, which could mean that LTL2Action essentially acts randomly given an infeasible task.

Effect of Edge-Matching Condition: We trained LTL-Transfer on *Mixed* training sets of varying sizes and tested zero-shot transfer on all five test sets. The success rates of using the *Relaxed* and *Constrained* edge-matching conditions are depicted in Figure 3d and Figure 3c, respectively. We observed that LTL-Transfer using the *Relaxed* edge-matching condition successfully transferred to significantly more novel specifications across all types, thus supporting **H2**.

Relative Difficulty of Specification Types: Figure 3c shows that LTL-Transfer with the *Relaxed* edge-matching condition can perfectly transfer to novel tasks of *Soft*, *Strictly-Soft* and *No-Orders* types after training on very few tasks. After training on 50 tasks, LTL-Transfer can transfer to over 95% of novel tasks of the *Mixed* type. Tasks of the *Hard* type are the most difficult to transfer to. Figure 3d shows that the different tasks are equally difficult to transfer to using the *Constrained* edge-matching condition. Thus, **H3** is supported only by using the *Relaxed* edge-matching condition.

Stochastic Environments: To evaluate the robustness of LTL-Transfer to stochastic transitions, we increased the slip probability from 0 to 0.4 and observed a slight 3% decrease in transfer success in Figure 3e, which supports **H4**.

D. Robot Demonstrations

We deployed LTL-Transfer at the task-planning level of a quadruped mobile manipulator, Spot [34], with off-the-shelf motion planning and grasping capabilities in a discretized indoor environment where the robot can fetch and deliver objects while navigating through the space. LTL-Transfer first learned policies from 20 training tasks, then transferred the learned skills to 100 novel tasks from the five specification types defined in Section VI-B, 50 of which were executed on the robot [1]. The environment contains 31 grid cells, but LTL-Transfer works in larger domains, like 19×19, as presented in Section VI-A. The state space includes the locations of the robot and six landmarks, i.e., two desks, a couch, a door, a bookshelf with a book, and a kitchen counter with a juice bottle on top. Actions are moving in the four cardinal directions. Only navigation skills are learned from the training tasks. Picking is executed when the goal location of a navigation option is the bookshelf or the counter. Placement is executed after the robot reaches the desk or the couch and has picked up an object.

VII. CONCLUSIONS

We introduced LTL-Transfer, a zero-shot transfer algorithm that uses the compositionality of LTL task specification to maximally transfer learned policies to solve various novel tasks. Experiments in deterministic and stochastic Minecraft-inspired domains showed that LTL-Transfer can solve complex unseen tasks without violating any safety constraints. We deployed LTL-Transfer at the task-planning level of a mobile manipulator to safely solve fetch-and-deliver and navigation tasks in zero-shot. We envision incorporating long-term planning and intra-option policy updates to produce not just satisfying but optimal solutions to novel tasks.

ACKNOWLEDGMENT

The authors thank Peilin Yu and Mingxi Jia for editing the videos. This work is supported by ONR under grant numbers N00014-21-1-2584, N00014-17-1-2699, and N00014-22-1-2592, NSF under grant number CNS-2038897, Amazon Robotics under award number 1061079, and funding from Echo Labs. Partial funding for this work was provided by The Boston Dynamics AI Institute ("The AI Institute").

[1]Videos are at `https://jasonxyliu.github.io/LTL-Transfer`

REFERENCES

[1] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

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

[3] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan, "Environment-independent task specifications via gltl," *arXiv preprint arXiv:1704.04341*, 2017.

[4] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018, pp. 452–461.

[5] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Ltl and beyond: Formal languages for reward function specification in reinforcement learning." in *IJCAI*, vol. 19, 2019, pp. 6065–6073.

[6] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah, "Grounding complex natural language commands for temporal tasks in unseen environments," in *Conference on Robot Learning (CoRL)*. PMLR, 2023, pp. 1084–1110.

[7] R. Patel, E. Pavlick, and S. Tellex, "Grounding language to non-markovian tasks with no supervision of task specifications," in *Robotics: Science and Systems*, 2020.

[8] A. Shah, P. Kamath, J. A. Shah, and S. Li, "Bayesian inference of temporal task specifications from demonstrations," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[9] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," *Logics for concurrency*, pp. 238–266, 1996.

[10] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *International Conference on Protocol Specification, Testing and Verification*. Springer, 1995, pp. 3–18.

[11] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal methods in system design*, vol. 19, no. 3, pp. 291–314, 2001.

[12] Z. Manna and A. Pneuli, "A hierarchy of temporal properties," in *ACM symposium on Principles of distributed computing*, 1990.

[13] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.

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

[15] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370299000521

[16] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur, "Compositional reinforcement learning from logical specifications," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[17] B. G. León, M. Shanahan, and F. Belardinelli, "Systematic generalisation through task temporal logic and deep reinforcement learning," *Adaptive and Learning Agents Workshop at International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[18] B. G. Leon, M. Shanahan, and F. Belardinelli, "In a nutshell, the human asked for this: Latent goals for following temporal specifications," in *International Conference on Learning Representations (ICLR)*, 2022.

[19] B. Araki, X. Li, K. Vodrahalli, J. Decastro, M. Fry, and D. Rus, "The logical options framework," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 307–317. [Online]. Available: https://proceedings.mlr.press/v139/araki21a.html

[20] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *International Conference on Machine Learning*. PMLR, 2017, pp. 166–175.

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

[22] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith, "LTL2action: Generalizing LTL instructions for multi-task RL," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 497–10 508.

[23] Z. Xu and U. Topcu, "Transfer of temporal logic formulas in reinforcement learning," in *IJCAI: proceedings of the conference*, vol. 28. NIH Public Access, 2019, p. 4010.

[24] G. N. Tasse, D. Jarvis, S. James, and B. Rosman, "Skill machines: Temporal logic composition in reinforcement learning," *arXiv preprint arXiv:2205.12532*, 2022.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[26] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et al.*, "Model-based reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[27] G. D. Konidaris and A. G. Barto, "Building portable options: Skill transfer in reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 895–900.

[28] S. James, B. Rosman, and G. Konidaris, "Learning portable representations for high-level planning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4682–4691.

[29] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *International Conference on Learning Representations*, 2019.

[30] A. Bagaria, J. Senthil, M. Slivinski, and G. Konidaris, "Robustly learning composable options in deep reinforcement learning," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, 2021.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[32] L. G. Valiant, "The complexity of computing the permanent," *Theoretical computer science*, vol. 8, no. 2, pp. 189–201, 1979.

[33] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[34] Boston Dynamics, "Spot® - the agile mobile robot," https://www.bostondynamics.com/products/spot.

APPENDIX I
LPOPL

A DQN policy is a feedforward network with two hidden layers, each of which has 64 ReLU units. We used the same hyperparameters suggested in [4] for training, i.e., the learning rate is 0.0001; the size of the replay buffer is 25,000; 32 transitions are randomly sampled from the replay buffer for every update; the discount factor is 0.9; exploration decreases linearly from 1 to 0.02.

APPENDIX II
EXAMPLE LTL FORMULAS

We provide LTL task specifications and their interpretations from the *Hard*, *Soft*, *Strictly Soft*, *No Orders*, and *Mixed* formula types.

Hard: Example formulas and their interpretations from the *Hard* type are as follows:

1) $\mathbf{F}workbench \wedge \mathbf{F}factory \wedge \mathbf{F}iron \wedge \mathbf{F}shelter \wedge \neg factory \ \mathbf{U} \ axe$: Visit $workbench$, $factory$, $iron$, $shelter$, and axe. Ensure that $factory$ is not visited before axe.
2) $\mathbf{F}toolshed \wedge \mathbf{F}bridge \wedge \mathbf{F}factory \wedge \mathbf{F}axe \wedge \neg bridge \ \mathbf{U} \ wood$: Visit $toolshed$, $bridge$, $factory$, axe, and $wood$. Ensure that $bridge$ is not visited before $wood$.
3) $\mathbf{F}wood \wedge \mathbf{F}axe \wedge \neg wood \ \mathbf{U} \ grass \wedge \neg grass \ \mathbf{U} \ workbench \wedge \neg workbench \ \mathbf{U} \ bridge$: Visit $bridge$, $workbench$, $grass$, $wood$, and axe. Ensure visiting $bridge$, $workbench$, $grass$, and $wood$ in that particular order. Objects that occur later in the sequence cannot be visited before any prior objects.

Soft: Example formulas and their interpretations from the *Soft* type are as follows:

1) $\mathbf{F}(bridge \wedge \mathbf{F}(factory \wedge \mathbf{F}(iron \wedge \mathbf{F}shelter)))$: Visit $bridge$, $factory$, $iron$, and $shelter$ in that sequence. The objects that occur later in the sequence may be visited before the prior objects, provided that they are visited at least once after the prior object has been visited.
2) $\mathbf{F}workbench \wedge \mathbf{F}(factory \wedge \mathbf{F}grass)$: Visit the $workbench$, $factory$, and $grass$: Visit $grass$ at least once after visiting the $factory$.
3) $\mathbf{F}(axe \wedge \mathbf{F}factory) \wedge \mathbf{F}workbench$: Visit axe, $factory$, and $workbench$. Ensure that $factory$ is visited at least once after axe.

Strictly Soft: Example formulas and their interpretations from the *Strictly Soft* type are identical to the *Soft* specifications, except they do not allow simultaneous satisfaction of multiple sub-tasks. The subtasks in the sequence must occur strictly after the prior subtask. This is enforced using nested operators next and finally $\mathbf{XF}a$ instead of $\mathbf{F}a$.

No Orders: These specifications only contain a list of subtasks to be completed. No temporal orders are enforced between any two subtasks.

1) $\mathbf{F}wood \wedge \mathbf{F}grass \wedge \mathbf{F}stone$: Visit $bridge$: Collect $wood$, $grass$, $stone$ in no particular order.

Mixed: Example formulas and their interpretations from the *Mixed* type are as follows:

1) $\mathbf{F}toolshed \wedge \mathbf{F}factory \wedge \neg toolshed \ \mathbf{U} \ shelter \wedge \mathbf{F}(grass \wedge \mathbf{F}bridge)$: Visit the $toolshed$, $factory$, $shelter$, $grass$, and $bridge$. Ensure that $toolshed$ is not visited before the $shelter$ and $bridge$ is visited at least once after $grass$.
2) $\mathbf{F}grass \wedge \neg grass \ \mathbf{U} \ toolshed \wedge \mathbf{F}(factory \wedge \mathbf{XF}workbench)$: Visit $grass$, $toolshed$, $factory$, and $workbench$. Ensure that $grass$ is not visited before $toolshed$ and $workbench$ is visited at least once strictly after $factory$.
3) $\mathbf{F}iron \wedge \neg iron \ \mathbf{U} \ toolshed \wedge \mathbf{F}(shelter \wedge \mathbf{XF}wood)$: Visit $iron$, $toolshed$, $shelter$, and $wood$. Ensure that $iron$ is not visited before $toolshed$ and $wood$ is visited at least once strictly after $shelter$.

APPENDIX III
ADDITIONAL EXPERIMENTAL RESULTS

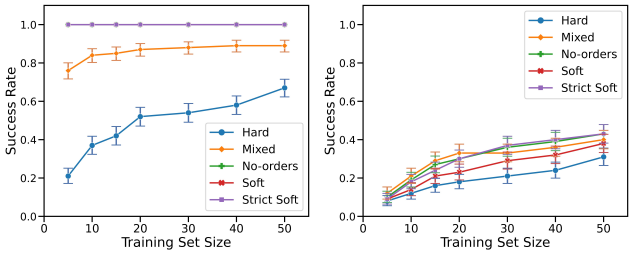
Learning Curves for Various Training Sets: We present learning curves of success rates for transferring policies learned on different specification types.

The learning curves for training on LTL tasks from the *Hard* training set with both *Relaxed* and *Constrained* edge-matching conditions are depicted in Figure 1.

The learning curves for training on LTL tasks from the *Soft* training set with both *Relaxed* and *Constrained* edge-matching conditions are depicted in Figure 2.

The learning curves for training on LTL tasks from the *Strictly Soft* training set with both *Relaxed* and *Constrained* edge-matching conditions are depicted in Figure 3.

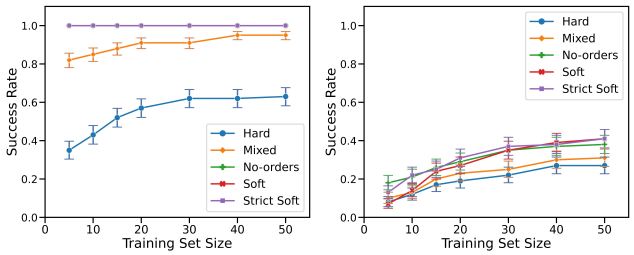
The learning curves for training on LTL tasks from the *No Orders* training set are being generated at the time of submission and are expected to share nearly identical trends as the learning curves from the other training sets. We will include the plots in the final version of the paper.



(a) *Relaxed* Edge Match (b) *Constrained* Edge Match

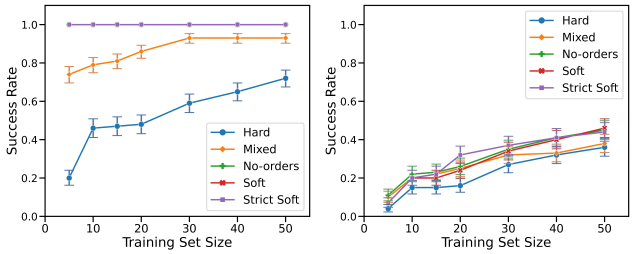
Fig. 1: Figure 1a depicts the success rates of transferring to five different specifications types using the *Relaxed* edge-matching condition after LTL-Transfer being trained on *Hard* training sets of various sizes. Figure 1b depicts the success rates with the *Constrained* edge-matching condition. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.

Note that for training on each specification type, the learning curve trends are nearly identical to the learning



(a) *Relaxed* Edge Match (b) *Constrained* Edge Match

Fig. 2: Figure 2a depicts the success rates of transferring to five different specifications types using the *Relaxed* edge-matching condition after LTL-Transfer being trained on *Soft* training sets of various sizes. Figure 2b depicts the success rates with the *Constrained* edge-matching condition. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.



(a) *Relaxed* Edge Match (b) *Constrained* Edge Match

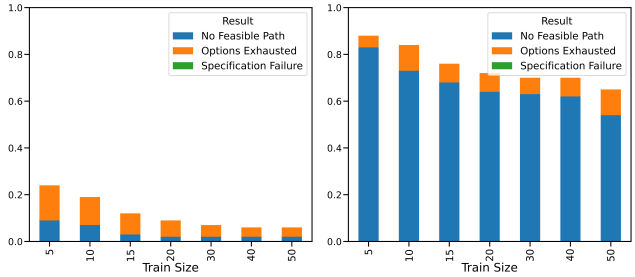
Fig. 3: Figure 3a depicts the success rates of transferring to five different specifications types using the *Relaxed* edge-matching condition after LTL-Transfer being trained on *Strictly Soft* training sets of various sizes. Figure 3b depicts the success rates with the *Constrained* edge-matching condition. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.

curves of training on the *Mixed* specification types, as depicted in Figure 3 in the main paper. *Hard* specification types remain the most challenging specification types to transfer to.

Failure Analysis: As described in the main paper, we logged the reason for the failure of each unsuccessful transfer attempt. There are three possible causes:

1) *Specification failure*: a constraint is violated during execution, and the reward machine progresses to an unrecoverable state.
2) *No feasible path*: there are no paths connecting the start reward machine state to an accepting reward machine state with matching transition-centric options.
3) *Options exhausted*: there are no further transition-centric options available to further progress the state of the reward machine.

Figure 4 depicts the relative frequency of the failure modes when LTL-Transfer is trained and tested on *mixed* task specifications. Note that with the *Relaxed* edge-matching condition not progressing the task after utilizing all available safe options is the primary reason for failure (Figure 4a) whereas, with the *Constrained* edge-matching condition, the absence of feasible paths connecting the start and the accepting state is the primary reason for failure (Figure 4b).



(a) *Relaxed* Edge Match (b) *Constrained* Edge Match

Fig. 4: Reasons for failed task execution after being trained and evaluated on the *Mixed* task specification datasets. Note that all values are depicted in fractions.

Appendix IV
Selected Solution Trajectories in Simulation

Consider the case with *Mixed* training set with 5 formulas on *Map 0*. The training formulas are:

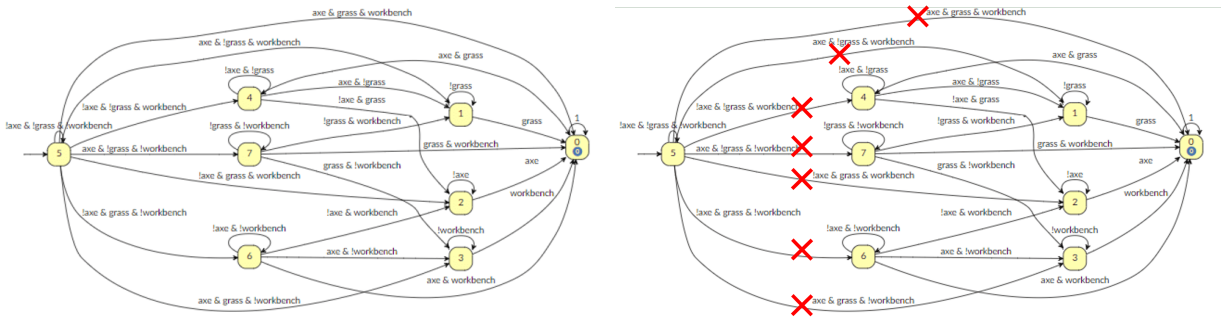
- $\mathbf{F}grass \wedge \mathbf{F}shelter \wedge \mathbf{F}(wood \wedge \mathbf{XF}workbench)$
- $\mathbf{F}toolshed \wedge \mathbf{F}workbench \wedge \mathbf{F}shelter \wedge (\neg toolshed\ \mathbf{U}\ shelter) \wedge \mathbf{F}(grass \wedge \mathbf{F}bridge)$
- $\mathbf{F}toolshed \wedge \mathbf{F}(shelter \wedge \mathbf{F}(axe \wedge \mathbf{F}wood))$
- $\mathbf{F}iron \wedge \mathbf{F}(shelter \wedge \mathbf{XF}(bridge \wedge \mathbf{XF}factory))$
- $\mathbf{F}factory$

One of the *Mixed* test formulas is $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$. The reward machine for this task specification is depicted in Figure 5a. Given the training set of formulas and the use of the *Constrained* edge-matching condition, the start reward machine state is disconnected from all downstream states as no transition-centric options match the edge transitions. Therefore, LTL-Transfer does not attempt to solve the task and returns failure with the reason being *no feasible path*, i.e., a disconnected reward machine graph after removing infeasible edges.

If the *Relaxed* edge-matching condition is used, there are matching transition-centric options for each edge. The trajectory adopted by LTL-Transfer when transferring the policies is depicted in Figure 6. The robot collects all three requisite resources before it terminates the task execution. Further, note that the robot passes through a grid containing $wood$ as the specification does not explicitly prohibit it.

Appendix V
Robot Demonstration

The 50 test tasks executed on the robot are shown in Table I. The Proposition a represents a brown desk, b represents a white desk, c represents a couch, d represents a door, s represents a bookshelf, and k represents a kitchen counter.



(a) (b)

Fig. 5: Figure 5a depicts the reward machine for the task specification $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$, as well as all feasible edges matched by the *Relaxed* condition. Note that all the edges have at least one matching transition-centric option for the *Relaxed* edge-matching condition. Figure 5b depicts the edges that do not have a compatible transition-centric option for the *Constrained* edge-matching condition.

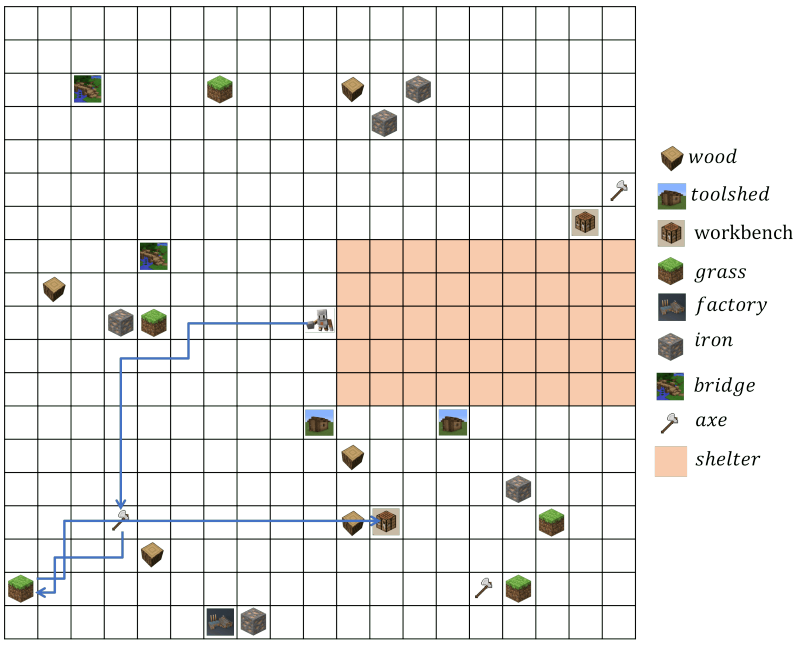


Fig. 6: Trajectory executed by the robot using LTL-Transfer to achieve the novel task specification $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$.

TABLE I: Test Tasks Executed on the Robot

LTL Task Specification	Type of Task	Results
0. $\mathbf{F}a$	navigation	success
1. $\mathbf{F}a \wedge \mathbf{F}b$	navigation	success
2. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}c$	navigation	success
3. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}s$	navigation	success
4. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}k$	navigation	success
5. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}c \wedge \mathbf{F}d$	navigation	success
6. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}c \wedge \mathbf{F}s$	navigation	success
7. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}c \wedge \mathbf{F}k$	navigation	success
8. $\mathbf{F}a \wedge \mathbf{F}b \wedge \mathbf{F}c \wedge \mathbf{F}k \wedge \mathbf{F}s$	navigation	success
9. $\mathbf{F}(b \wedge \mathbf{F}(a \wedge \mathbf{F}(c \wedge \mathbf{F}d))))$	navigation	success
10. $\mathbf{F}(s \wedge \mathbf{F}a)$	fetch and deliver	success
11. $\mathbf{F}(s \wedge \mathbf{F}b)$	fetch and deliver	success
12. $\mathbf{F}(a \wedge \mathbf{F}b)$	navigation	success
13. $\mathbf{F}(b \wedge \mathbf{F}a)$	navigation	success
14. $\mathbf{F}(a \wedge \mathbf{F}(s \wedge \mathbf{F}c))$	fetch and deliver	success
15. $\mathbf{F}(b \wedge \mathbf{F}(s \wedge \mathbf{F}c))$	fetch and deliver	success
16. $\mathbf{F}(s \wedge \mathbf{F}(a \wedge \mathbf{F}c))$	fetch and deliver	success
17. $\mathbf{F}(a \wedge \mathbf{F}(b \wedge \mathbf{F}c))$	navigation	success
18. $\mathbf{F}(s \wedge \mathbf{F}(a \wedge \mathbf{F}(k \wedge \mathbf{F}a))))$	fetch and deliver	success
19. $\mathbf{F}(a \wedge \mathbf{F}(b \wedge \mathbf{F}(c \wedge \mathbf{F}d))))$	navigation	success
20. $\mathbf{F}(s \wedge \mathbf{XF}a)$	fetch and deliver	success
21. $\mathbf{F}(b \wedge \mathbf{XF}s)$	navigation	success
22. $\mathbf{F}(a \wedge \mathbf{XF}b)$	navigation	success
23. $\mathbf{F}(b \wedge \mathbf{XF}a)$	navigation	success
24. $\mathbf{F}(a \wedge \mathbf{XF}(b \wedge \mathbf{XF}c))$	navigation	success
25. $\mathbf{F}(a \wedge \mathbf{XF}(s \wedge \mathbf{XF}b))$	fetch and deliver	success
26. $\mathbf{F}(b \wedge \mathbf{XF}(s \wedge \mathbf{XF}a))$	fetch and deliver	success
27. $\mathbf{F}(s \wedge \mathbf{XF}(b \wedge \mathbf{XF}a))$	fetch and deliver	success
28. $\mathbf{F}(k \wedge \mathbf{XF}b)$	fetch and deliver	success
29. $\mathbf{F}(k \wedge \mathbf{XF}a)$	fetch and deliver	success
30. $\neg a \mathbf{U} s \wedge \mathbf{F}a$	fetch and deliver	success
31. $\neg b \mathbf{U} a \wedge \mathbf{F}b$	navigation	success
32. $\neg a \mathbf{U} b \wedge \mathbf{F}a$	navigation	success
33. $b \mathbf{U} a \wedge \neg c \mathbf{U} b \wedge \mathbf{F}c$	navigation	success
34. $b \mathbf{U} k \wedge \mathbf{F}b$	fetch and deliver	success
35. $\neg b \mathbf{U} c \wedge \mathbf{F}b$	navigation	success
36. $\neg a \mathbf{U} s \wedge \neg b \mathbf{U} a \wedge \mathbf{F}b$	fetch and deliver	success
37. $\neg s \mathbf{U} a \wedge \neg b \mathbf{U} s \wedge \mathbf{F}b$	fetch and deliver	success
38. $\neg b \mathbf{U} a \wedge \neg s \mathbf{U} b \wedge \mathbf{F}s$	navigation	success
39. $\neg a \mathbf{U} b \wedge \neg s \mathbf{U} a \wedge \mathbf{F}s$	navigation	success
40. $\mathbf{F}a \wedge \mathbf{F}(b \wedge \mathbf{F}c)$	navigation	success
41. $\mathbf{F}a \wedge \neg c \mathbf{U} b \wedge \mathbf{F}c$	navigation	success
42. $\mathbf{F}(a \wedge \mathbf{F}b) \wedge \neg c \mathbf{U} a \wedge \mathbf{F}c$	navigation	success
43. $\mathbf{F}a \wedge \mathbf{F}(b \wedge \mathbf{XF}c)$	navigation	success
44. $\mathbf{F}(a \wedge \mathbf{F}b) \wedge \neg c \mathbf{U} b \wedge \mathbf{F}c$	navigation	success
45. $\mathbf{F}(b \wedge \mathbf{F}a) \wedge \neg c \mathbf{U} b \wedge \mathbf{F}c$	navigation	success
46. $\mathbf{F}c \wedge (\neg s \mathbf{U} a \wedge \mathbf{F}s)$	navigation	success
47. $\mathbf{F}c \wedge (\neg a \mathbf{U} s \wedge \mathbf{F}a)$	navigation	success
48. $\mathbf{F}c \wedge (\neg s \mathbf{U} b \wedge \mathbf{F}s)$	navigation	success
49. $\mathbf{F}c \wedge (\neg b \mathbf{U} s \wedge \mathbf{F}b)$	navigation	success