
CHALMERS
UNIVERSITY OF TECHNOLOGY

# Enhancing Hierarchical Reinforcement Learning with Symbolic Planning for Long-Horizon Tasks



Citation for the original published paper (version of record):
Zhang, J., Dean, E., Ramirez-Amaro, K. (2026). Enhancing Hierarchical Reinforcement Learning with Symbolic Planning for Long-Horizon Tasks. IEEE Transactions on Automation Science and Engineering, 23: 3169-3184. http://dx.doi.org/10.1109/TASE.2025.3641255

# Enhancing Hierarchical Reinforcement Learning With Symbolic Planning for Long-Horizon Tasks

Jing Zhang, *Student Member, IEEE*, Emmanuel Dean, *Member, IEEE*, and Karinne Ramirez-Amaro, *Member, IEEE*

***Abstract*—Long-horizon tasks, such as box stacking, pose a longstanding challenge in robotic manipulation, especially for Reinforcement Learning (RL). RL typically focuses on learning an optimal policy for completing an entire task, rather than determining the specific sequence of actions required to achieve complex goals. While RL finds a sequence of actions that maximizes the total reward of the task, the main challenge arises when there are infinite possibilities of chaining actions (e.g. `reach`, `push`) to achieve the same task (door-opening). In this case, RL struggles to find the optimal policy. In contrast, symbolic planning focuses on determining a sequence of actions to achieve the desired task. This paper introduces a novel framework that integrates symbolic planning operators with hierarchical RL. We propose to change the way complex tasks are trained by learning independent policies for actions defined by high-level operators instead of learning a single policy for the complete long-horizon task. Our approach easily adapts to various tasks by adjusting the learned operator set on demand. We developed a dual-purpose high-level operator, which can be used both in holistic planning and as independent, reusable policies. Our approach offers a flexible solution for long-horizon tasks, e.g., stacking and inserting a cube, and door-opening. Experimental results indicate that our method achieves high success rates of around 95% in policy chaining for comprehensive plan execution and excels in learning independent policies. Furthermore, it remains robust and scalable even with a small sample set evaluation, attaining an 84% success rate for planning and executing a new task (door-opening) and 85% when dealing with a new operator. Experiments in dynamic environments further demonstrate the robustness and adaptability of our approach, which sustains a high success rate of around 90% and outperforms all baselines.**

***Note to Practitioners*—Tasks requiring multiple steps are defined as long-horizon tasks, e.g., stacking, inserting, and door opening, remain challenging to deploy in settings such as manufacturing, logistics, and service robotics. Conventional approaches predominantly depend on task-specific engineering paradigms: either laborious manual policy scripting requiring explicit kinematic/dynamic modelling, or data-intensive imitation learning frameworks demanding thousands of task-specific demonstrations. These methodologies exhibit critical limitations in scalability and environmental adaptability, particularly when confronted with novel objects, workspace configurations, or task variants. Our work proposes an alternative by breaking complex tasks into smaller and independent sub-tasks, each capturing a distinct behaviour (e.g., `reach`, `lift`, `push`). Moreover, our proposed approach demonstrates how to sequence sub-tasks to accomplish complex, long-horizon tasks. During this process, the sub-tasks are learned as operators using concepts from symbolic planning methods. The main benefit of such operators is that they can be reused; therefore, adding new tasks requires less re-engineering. Four types of dynamic environment experiments further demonstrate the robustness and adaptability of our approach.**

***Index Terms*—Hierarchical reinforcement learning, symbolic planning, long-horizon task.**

## I. Introduction

LONG horizon tasks, such as stacking cubes or assembling products, present a particularly daunting challenge for autonomous systems. Autonomous agents must learn the intricate dependencies between *actions*, understanding how each action influences subsequent steps (*effects*), how actions should be executed, and ultimately how to reach the desired goal. The core motivation of our work is to develop methods that can effectively learn and execute long-horizon tasks. Existing approaches, such as behavioural cloning (BC) [1], [2], offer an intuitive approach by mimicking human actions; they heavily rely on either extensive manual coding or large volumes of demonstration data, both of which hinder scalability and adaptability. Deep Reinforcement Learning (DRL) has emerged as a promising paradigm for learning control policies through interaction with the environment. The agent iteratively improves its decision-making to maximize cumulative rewards over time. However, the effectiveness of DRL in long-horizon tasks is limited by the probabilistic nature of state transitions. Random exploration is insufficient for long-horizon tasks, as the main challenge lies in task sequencing. Early actions can have long-term consequences, and accumulated uncertainty over time may result in suboptimal outcomes or task failure.

Learning a sequence of actions is challenging, and the symbolic planning methods address this problem. Planners serve as a structured and scalable framework for tackling complex, long-horizon operations. These approaches enhance interpretability, as humans typically provide the foundational knowledge for symbolic structures.

Received 12 February 2025; revised 15 July 2025 and 9 October 2025; accepted 20 November 2025. Date of publication 8 December 2025; date of current version 9 February 2026. This article was recommended for publication by Associate Editor P. Scarabaggio and Editor C. Seatzu upon evaluation of the reviewers' comments. This work was supported in part by the Chalmers AI Research Centre (CHAIR); and in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. *(Corresponding author: Jing Zhang.)*

The authors are with the Faculty of Electrical Engineering, Chalmers University of Technology, 412 96 Gothenburg, Sweden (e-mail: jing.zhang@chalmers.se; deane@chalmers.se; karinne@chalmers.se).

This article has supplementary downloadable material available at https://doi.org/10.1109/TASE.2025.3641255, provided by the authors.

Digital Object Identifier 10.1109/TASE.2025.3641255

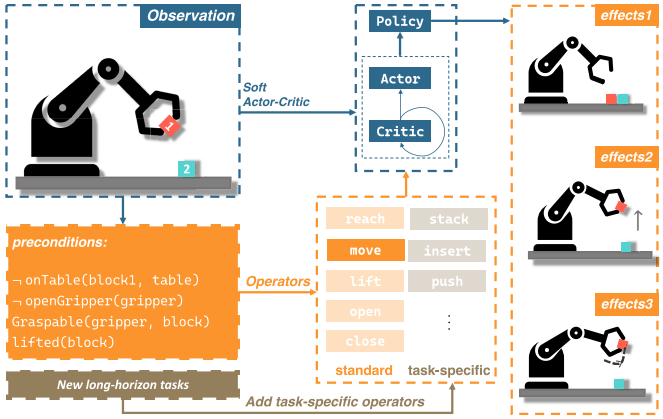


Fig. 1. Overview of our approach: We combine reinforcement learning (RL) with planning operators, allowing for the seamless integration of novel operators to adapt to evolving tasks.

Symbolic planning involves two main components: a *domain model* and a *problem description*. A *domain model* is a formal description of the task environment that defines the types of objects, predicates, and actions, including their preconditions and effects that govern which operators[1] are available for execution. The *problem description* specifies the initial state and the goal. However, a key limitation lies in the necessity for an exhaustive and accurate delineation of the *domain model*. The manual construction of such comprehensive domains is often impractical or even unfeasible, as pointed out in [3]. Moreover, an external planner/task solver is required to select appropriate operators and their sequence to achieve the task's goal. Therefore, the challenge remains to find an efficient, universally applicable solution that is both adaptable and scalable for diverse long-horizon task executions.

In this paper, we will address the following challenges:

1) Adapting the classical approach in Reinforcement Learning by decomposing complex tasks into manageable sub-tasks.
2) Developing an automatic sequencing method to effectively chain learned independent policies.
3) Integrating symbolic planning and policy execution into a unified, holistic learning framework (see Figure 1).

Particularly, we use the definition of operators from the symbolic planning domain [4] to enhance the construction of the action hierarchy used in the hierarchical RL (HRL) method based on the Schedule Auxiliary Control framework (SAC-X) [5]. The SAC-X method randomly chooses tasks to achieve the desired goal/task. In our proposed method, first, we extract the environmental state and encode its dynamics as preconditions and effects. These state-based observations are used to define different planning operators, which in turn automatically generate a sequence of actions that the agent should execute to achieve the desired goal. This information is then used by the RL agent to independently learn and execute low-level policies, where each policy is trained separately from the others.

We propose an adaptive mechanism that enables dynamic operator selection through real-time evaluation of environmental states. Context-aware operators can be reused or flexibly extended when new long-horizon tasks are requested, requiring only the addition of new operators if novel interactions are introduced. In this work, we define independent operators, meaning that each operator will learn its own independent low-level control policy. The goal is to learn policies that can be used even without a planning process (e.g., PDDL). Low-level policies are responsible for executing specific manipulator translational movements in a physical context. With our proposed learning approach, robots can learn the optimal execution sequence of high-level operators[2] for accomplishing long-horizon tasks while implementing these operators through low-level manipulator policies. We validated our learning method in three different long-horizon scenarios (cube stacking, cube insertion, and door-opening) and six independent task polices (`reach`, `lift`, `move`, `stack`, `insert`, `push`). Furthermore, the door-opening task demonstrates our method's scalability in integrating new operators, showcasing its adaptability to a broader range of tasks.

Previous RL approaches on robotic manipulation have typically focused on static environments, often overlooking dynamics and failing to evaluate whether policies trained under static conditions remain robust and adaptable when exposed to real-time dynamic movements and perturbations [6]. To bridge the gap, we designed four dynamic environment tests to systematically evaluate the robustness of our approach. The experimental results highlight the novelty and significance of our contribution to long-horizon manipulation tasks.

In summary, our main contributions are:

1) We introduce a novel learning framework that combines hierarchical Reinforcement Learning and symbolic planning to automatically generate action sequences based on environment observations in the form of preconditions and effects.
2) We design an on-demand high-level decision-making system capable of chaining independent policies to achieve complex tasks, while adapting to dynamic environmental changes.
3) We develop a method that jointly learns low-level, independent policies and enables their automatic sequencing through the high-level system, facilitating the holistic execution of long-horizon tasks in dynamic environments.

Furthermore, we evaluated our approach across various scenarios, considering both static and dynamic environments.

## II. Related Works

To tackle long-horizon tasks, robots must be equipped with the ability to anticipate future states. This implies that an

[1]Operators are also known as actions, here we use the term operators to discriminate from the action concepts used in RL. In this context, operators are abstract descriptions that can contain multiple actions to achieve their execution.

[2]In our method, the high-level system makes decisions on-demand based on interactions with the environment, thereby obviating reliance on pre-defined plans or external planners (e.g., PDDL-based methods).

abstract representation, including the environment state, the robot's capabilities and strategies, and the task objectives, is essential, since a key feature of intelligent behavior is the ability to learn abstract strategies that can scale and transfer to unfamiliar problems [7]. Our research draws inspiration from the different areas of reinforcement learning and symbolic planning.

### A. Symbolic Approaches in Manipulation

Symbolic planning is central to the aforementioned abstraction mechanism in robot learning, which has been extensively explored over many years [8]. This abstraction paradigm involves representing the world as a set of discrete symbols (e.g., predicates, actions, objects) and using logic-based methods to determine how those symbols can change from an initial to a goal state. STRIPS [9] and Planning Domain Definition Language (PDDL) [10], [11] are formalisms used within this paradigm. While they demonstrated early success in deterministic domains, they heavily rely on strictly engineered symbolic representations and specialized external solvers, posing significant scalability challenges [3]. Moreover, these methods focus on high-level action sequencing without directly addressing low-level control execution, limiting their practicality for real-world robotic tasks.

Task and Motion Planning (TAMP) [12] can be categorized by: (1) sampling-based TAMP, which integrates symbolic reasoning such as PDDL or Behaviour Tree (BT) with continuous motion planning methods, (2) optimization-based TAMP, which adheres to constraints imposed by the robot kinematics and dynamics at the motion planning level. In Logic-geometric programming (LGP) [13], [14], the entire motion planning is optimized based on geometric constraints, which are transformed into subtasks. These subtasks require manual specification [15], and the skill library is pre-trained. Styrud et al. [16] employ Bayesian optimization to fine-tune the parameters of a behavior tree; a planner is used to obtain the BT structure. In [15], the authors proposed a hierarchical planning architecture that integrates an LGP subtask planner, a TAMP solver, and a subtask scheduler to enable real-time low-level replanning. However, the computation time required for logic refinement explodes as objects and predicates increase.

Several recent studies have sought to integrate symbolic reasoning with learning-based methods. Reference [17] used a transformer architecture to encode the progressed Linear Temporal Logic (LTL) formula as task latent features. They used *hard-coded* closed-loop controllers for each action. The work in [18] used an automaton as experience replay and policy gradient to learn LTL-satisfying policies, but has stability problems in LTL and has been tested only in a 2D grid world. Reward Machine (RM) is an increasingly popular method in task solving, but predominantly focuses on discrete low-dimensional state space [19], [20], [21], [22], [23]. The authors in [24] integrated a decision tree within a high-level policy and generated skill embeddings to guide the low-level policy execution. This approach relies on an offline, task-agnostic dataset and a pre-trained decision tree (DT), making extension to new tasks labor-intensive. The work [25] requires an additional PDDL planner, and no environment feedback is included. Therefore, it is best suited for static environments rather than dynamic ones.

### B. Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning enables the autonomous decomposition of challenging long-horizon decision-making tasks into simpler subtasks [26]. By decomposing tasks, an originally long-horizon process is transformed into a series of shorter-horizon subtasks. HRL algorithms have been demonstrated to outperform standard RL methods in several long-horizon challenges, including robot manipulation [27], [28], [29]. To learn the hierarchy, [30] uses natural language as the abstraction of a high-level planner, with the limitation that it requires predefined mappings between natural language instructions and tasks. Recent work has explored learning hierarchical policies from expert demonstrations [31], which hinders the extension of high-level policies to new tasks. The work in [32] introduces an asymmetric self-play mechanism in which the low-level policy learns a sub-goal representation space through self-play. Then, the high-level policy is trained on this representation, enabling it to generate appropriate sub-goal vectors to guide the low-level policy in accomplishing complex tasks. However, this approach relies on having a pre-trained self-play model. Riedmiller et al. [5] proposed the Scheduled Auxiliary Control (SAC-X) which learns low-level policies as auxiliary tasks and schedules them in search of sparse rewards of externally defined target tasks, but it struggles to learn long-horizon task and incurs high computational cost, as evidenced by the total $64.8\,M$ time steps required for learning a stacking task [33].

### C. Integrating Symbolic Planning With HRL

Symbolic planning could provide guidance and has proven to be a feasible method when combined with RL [34], [35]. These existing approaches have been effective in generating holistic plans for such tasks [36], [37]. However, they often encapsulate operators within the planning framework, making it infeasible to utilize these operators in isolation for other tasks or scenarios. Mehta and Zarrin [38] used imitation learning to study high-level planners in dexterous manipulation tasks, and a Teacher-Student RL framework for the low-level policies. The inherent structural constraints of the planner limit its scalability to more diverse tasks. The authors in [39] employed a Bayesian approach to select the optimal sub-goal since it requires human intervention to define both the candidate sub-goal space and the potential correlations among all sub-goals.

In contrast, our work provides a higher degree of modularity and flexibility. Specifically, our method not only integrates high-level operators derived from symbolic methods but also enables the generated high-level operators to function both within the scope of a specific plan and as separate, reusable policies. By incorporating the most recent environmental data, the agent dynamically selects the operator best suited for the current context. This design choice further facilitates

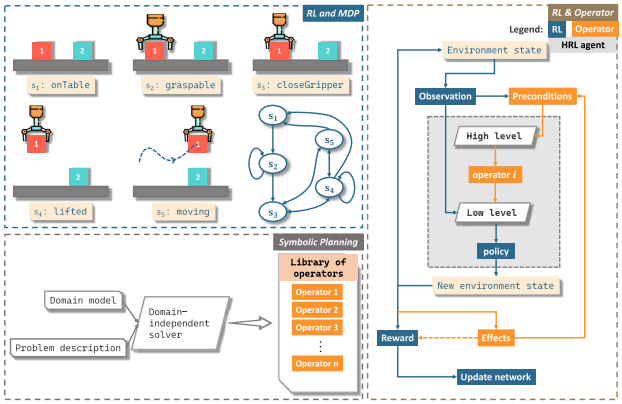


Fig. 2. The left figure shows the traditional RL and a symbolic planning framework. Here, $s$ describes the state of the cubes (*onTable, lifted*) and the gripper (*open* or *close*). The right figure illustrates our proposed framework combining RL and Operators.

straightforward scalability of the operator set to accommodate new tasks. The unique feature addresses the long-standing challenge in RL of learning the action sequences for long-horizon tasks while offering a greater range of applicability.

## III. Method

### A. Preliminaries

*1) Reinforcement Learning:* A Markov decision process (MDP) is defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma\}$, here $\mathcal{S}$ and $\mathcal{A}$ are the sets of state and actions, respectively. $\mathcal{P}$ is the state-transition kernel:

$$\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1], \tag{1}$$

and

$$r(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R} \tag{2}$$

is a possible reward function. $\gamma$ is the discount factor that ranges between 0 and 1. The solution of the MDP is a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ mapping a state to an action; actions are sampled from a policy $\pi(a|s)$. RL aims at maximizing the value function:

$$V^{\pi} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{t_f} \gamma^t \mathcal{R}\left(s_t, a_t\right)\right] \mid a_t \sim \pi\left(\cdot \mid s_t\right), s_{t+1} \sim \mathcal{P}\left(\cdot \mid s_t, a_t\right) \tag{3}$$

where $t$ refers to the time step and $t_f$ is the target time to achieve the goal or the period to complete a trajectory. The *RL and MDP* cube in Figure 2, shows a simple 5-state MDP. In long-horizon tasks, such as complex object manipulation in dynamic environments, RL algorithms face challenges in identifying the optimal sequence of actions to reach the goal. One of the primary reasons is the probabilistic nature of state transitions. The same action can lead to different subsequent states at different times. This cumulative uncertainty over a long sequence further complicates the learning process, e.g., in Figure 3, starting from $s_4 : lifted$, the next state could be $s_5$ or $s_1$, $s_4$, leading to optimal and not optimal plans, respectively. Since RL agents must explore various action sequences without guidance, and MDPs neither retain historical information nor detect execution failures, learning an optimal sequence

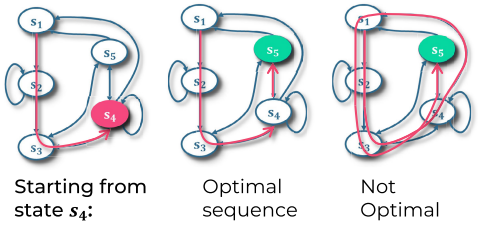


Fig. 3. Illustration of different subsequent states leading to optimal and non-optimal outcomes in long-horizon tasks. Starting from $s_4$, we can generate optimal and non-optimal sequences (red lines).

is challenging and may even fail. Thus, improving action sequencing in reinforcement learning is crucial. To address this challenge, we propose leveraging the expressiveness of high-level methods, such as symbolic planning, to facilitate action sequence learning.

*2) Symbolic Planning:* focuses on generating a sequence of high-level operators to achieve a specific goal. Symbolic planning employs symbolic language to define operators through preconditions and effects. Then, it reasons about these symbols to infer a sequence of actions using a planner algorithm. Such planners require additional information about the initial and goal states to obtain a plan. A planning task is typically divided into two main components: the *domain model* and the *problem description*. The *problem description* contains the initial condition and goal that needs to be reached while *domain model* offers a comprehensive list of all feasible operators that can be taken to transition from the initial state to the desired goal state. In this work, we define a list of operators necessary for learning and executing the tasks. This means that the observed environment is transformed into symbolic representations as a set of operators. One advantage of using the *domain model* is that it can be applied to multiple problems as long as they operate within the same set of fluents[3] and operators.

In symbolic planning, operators are templated action schematics that, when applicable, modify the state of the environment in a predetermined manner. A unique name identifies each planning operator and consists of three key elements:

i *Arguments*: a collection of objects that serve as parameters for the operator, specifying its context. For example, moving a cube can be represented by the operator `move`(cube), where cube is the argument.

ii *Preconditions*: these are logical conditions or criteria that must hold *True* in the current environment state for the operator to be applicable. e.g., in the example above, executing the operator `move`(cube) requires that certain preconditions be satisfied: the cube must be graspable; the cube is lifted and not on the table, etc.

iii *Effects*: these are the outcomes (or changes) that occur in the environment state after the execution of the operator. Similar to preconditions, effects are formally expressed to predict environmental transitions resulting from the application of an operator, e.g., once the

[3]Fluent is an instantiated predicate that describes relationships among objects in the environment.

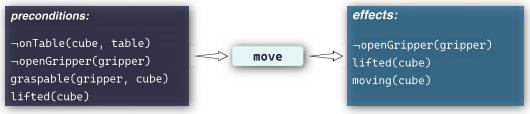


Fig. 4. Operator structure which is composed of preconditions and effects. In this example, onTable is a fluent with two input variables, *cube* and *table*, describing the state of objects. This fluent is *True* when *cube* is on the *table*.

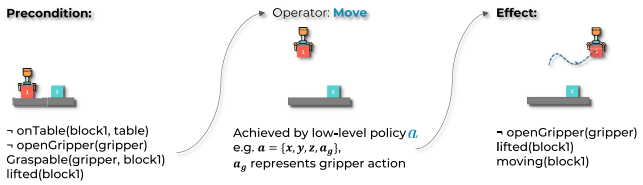


Fig. 5. How to split and chain policies based on preconditions and effects. Based on the *Precondition*, an *Operator* is selected, in this case *Move*. After executing the selected operator, we obtain the observed *Effect*.

operator `move`(cube) action has been executed, its effect is that the cube is now in motion (i.e., it acquires a non-zero velocity).

Both preconditions and effects use binary fluents, often referred to as predicates, which help to define the conditions and consequences in a structured manner. For example, the operator (`lift`) with its respective set of preconditions and effects is shown in Figure 4.

For the definition of *domain model*, we followed the convention of automated planning [4]. A *domain model* is defined as:

$$\Sigma = (\mathcal{F}, \mathcal{O}, \Gamma) \tag{4}$$

where:

- $\mathcal{F}$ is the set of propositional symbols referred to as fluents in the domain $\Sigma$. These fluents represent boolean properties of the environment.
- $\mathcal{O}$ is the set of operators available in the domain.
- $\Gamma : \mathcal{F} \times \mathcal{O}$ is the state-transition function.

Each fluent in $\mathcal{F}$ is an instantiated predicate that describes relationships among objects in the environment. As shown in Figure 2, the symbolic description of operators allows interpretability and transferability at a high level. However, the obtained operators do not contain information on low-level parametrization needed for the robot execution. For instance, consider a cube-stacking scenario where an operator is defined as `move`. This operator does not contain the specifics of how the manipulator should move the target object, for example, the required position, velocity, or orientation. This limitation highlights one of the challenges of applying symbolic planning definitions in scenarios that require low-level control details. Therefore, in this paper, we are proposing a solution for combining high-level operators and low-level control policies. In Figure 5, the precondition determines whether `move` is the appropriate choice for the next step based on the current environment state. The low-level policy ($a$) attempts to execute `move`. In our case, $a$ is a 4-dimensional vector, where the first three dimensions represent delta changes in positions relative to the end-effector frame, and the fourth dimension describes the gripper action—either open or close. Since the action $a$ changes the environment state, if `move` is successfully achieved, the current environment state will reflect the effects of the operator `move`, see Figure 5.

### B. Task Decomposition

*Definition 1 (Task Decomposition):* The task decomposition problem can be defined as:

$$\mathcal{D} = \{\langle \pi_{\mathcal{O}_i}, L(s^{\mathcal{O}_i}) \rangle\}_{i=1}^{K}, \quad \forall\ i,\ s^{\mathcal{O}_i} \triangleq (s_0^{\mathcal{O}_i}, s_T^{\mathcal{O}_i}) \tag{5}$$

We assume that a long-horizon task $\mathcal{T}$ can be decomposed into $K$ atomic sub-tasks, each corresponding to an operator $\{\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_K\}$. For each operator $\mathcal{O}_i$, let $\pi_{\mathcal{O}_i} : \mathcal{S} \to \mathcal{A}$ denote the corresponding low-level policy responsible for executing operator $\mathcal{O}_i$, where $\mathcal{A}$ is the action space and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function. Then, giving an initial state $s_0^{\mathcal{O}_i} \in \mathcal{S}$, the execution of $\pi_{\mathcal{O}_i}$ generates a trajectory:

$$\tau_{\mathcal{O}_i} = \{s_0^{\mathcal{O}_i}, s_1^{\mathcal{O}_i}, \cdots, s_T^{\mathcal{O}_i}\}, \quad \text{for } t = 0, 1, \cdots, T \tag{6}$$

$s_T^{\mathcal{O}_i}$ is the goal state of $\mathcal{O}_i$. To ensure proper sequencing, the execution of operator $\mathcal{O}_{i+1}$ is conditioned on the initial state $s_0^{\mathcal{O}_{i+1}}$ satisfied by the goal state achieved by the preceding operator $\mathcal{O}_i$, i.e.,

$$s_0^{\mathcal{O}_{i+1}} = s_T^{\mathcal{O}_i} \tag{7}$$

To bridge the gap between continuous state spaces $\mathcal{S}$ and their symbolic representations, which are required to select an operator, we introduce a translation function for grounding fluents:

$$L : \mathcal{S} \to 2^{\mathcal{F}}, \tag{8}$$

which maps a continuous state $s$ to a set of pre-defined fluents $\mathcal{F}$. The operator $\mathcal{O}_i$ is associated with a precondition $\phi(\mathcal{O}_i)$ and a effect $\psi(\mathcal{O}_i)$, denoted as:

$$\begin{aligned} \phi(\mathcal{O}_i) &\subseteq L(s_0^{\mathcal{O}_i}) \\ \psi(\mathcal{O}_i) &\subseteq L(s_T^{\mathcal{O}_i}) \end{aligned} \tag{9}$$

When operator $O_i$ is triggered, it induces changes in the environment state. The function $\psi(O_i)$ then verifies whether these expected changes occur by examining their associated fluents. Specifically, $\psi(O_i)$ does not capture the complete resulting state $s_T^{O_i}$ after successful execution of $O_i$; rather, it includes only the subset of fluents affected by $O_i$. Any fluent not included in $\psi(O_i)$, its value either remains unchanged or is not relevant to assessing the success of $O_i$. This is critical for scalability, because we only record the information relevant to $O_i$ rather than the entire state.

Given the condition:

$$\phi(\mathcal{O}_{i+1}) \subseteq \psi(\mathcal{O}_i), \quad \forall\ i \in \{1, 2, \cdots, K-1\}, \tag{10}$$

we ensure the seamless initiation of each operator upon completion of the preceding one.

TABLE I
DEFINITIONS OF THE FLUENTS AND THEIR RESPECTIVE GROUNDINGS

| fluent | Grounding |
|---|---|
| openGripper($g$) | binary action $a_g < 0$ |
| graspable($\alpha, g$) | $\|\|p_\alpha - p_g\|\| < 0.01$ |
| onTable($\alpha, \beta$) | $p_{\alpha,z} > p_{\beta,z} \wedge \ contact(\alpha, \beta)$ |
| lifted($\alpha$) | $p_{\alpha,z} > 0.01m$ |
| moving($\alpha$) | $\|\|vel_\alpha\|\| > 0.05m/s \wedge \|\|acc_\alpha\|\| < 5m/s^2$ |
| above($\alpha, \beta$) | $p_{\alpha,z} - p_{\beta,z} > 0.05m \wedge$ $\|\|(p_{\alpha,x}, p_{\alpha,y}) - (p_{\beta,x}, p_{\beta,y})\|\| \leqslant 0.01$ |
| onTop($\alpha, \beta$) | $p_{\alpha,z} - p_{\beta,z} > 0.035m \wedge \ contact(\alpha, \beta)$ |
| inSlot($\alpha, \beta$) | $\|\|p_\alpha - p_\beta\|\| < 0.003$ |
| rotated($\alpha$) | $\theta_\alpha > 20^\circ$ |

*1) Introducing New Operators:* A crucial challenge is systematically determining which operators are indispensable for effectively representing diverse long-horizon tasks. In Zanchettin's work [40], skill semantics and waypoints are extracted from kinesthetic demonstrations. Then, the system compares its state before and after each demonstration, using predefined fluents or predicates (as shown in Table II), to identify which predicates remain unchanged, are activated, or deactivated. This analysis provides insight into the underlying state transitions and the necessary semantic descriptors.

Inspired by that work, we selectively introduce new operators whenever the current operator set cannot capture the transitions required by the target task. Specifically, we assess completeness by comparing the symbolic state transitions needed by the task with the effects provided by the existing operators. Let $f \subseteq \mathcal{F}$ denote a symbolic state, where $\mathcal{F}$ is a set of fluents, and $\Omega$ denotes the current set of operators. Each operator $\mathcal{O}_i \in \Omega$ can be defined as $\mathcal{O}_i = (\phi(\mathcal{O}_i), \psi(\mathcal{O}_i))$ according to eq. 9. An operator $\mathcal{O}_i$ is applicable in a symbolic state $f$ if $\phi(\mathcal{O}_i) \subseteq f$. We now provide the definitions of *Cover* and *Task Completeness*:

*Definition 2 (Cover):* Given a transition $\tau = (f_{init}, f_1, \cdots, f_k, f_{goal})$, we say that "$\Omega$ covers $\tau$" (or $\tau$ is realizable by $\Omega$) iff. there exists a sequence of length $k$ consisting of $n$ operators from $\Omega$,

$$(\mathcal{O}_{i_1}, \mathcal{O}_{i_2} \cdots, \mathcal{O}_{i_k}), \quad i_l \in \{1, 2, \cdots, n\} \tag{11}$$

which results in $f_{init} \rightarrow f_{goal}$ after the following rules are executed. Formally:

1) $\phi(\mathcal{O}_{i_1}) \subseteq f_{init}$.
2) $\phi(\mathcal{O}_{i_{l+1}}) \subseteq f_l, l = 1, 2, \cdots, k-1$.
3) The final state $f_k$ after the last operator in the sequence: $f_k = f_{goal}$.

In this definition, transition $\tau$ encodes both the *desired* final fluent set $f_{\text{goal}}$ and the *actual* fluent set $f_k$ reached after executing a candidate operator sequence. By including them separately, we can distinguish successful realizations (where $f_k = f_{\text{goal}}$) from failed ones (where no sequence achieves that equality). This means that if no sequence of operators in $\Omega$ can achieve $f_{goal}$, then the transition $\tau$ is not covered by $\Omega$.

*Definition 3:* [Task Completeness]If $\forall \tau \in \mathcal{T}$, $\tau$ is covered by $\Omega$, then $\Omega$ is complete for $\mathcal{T}$; otherwise, it is incomplete, i.e.,

$$\text{Cover}(\Omega, \tau) = \begin{cases} 1, & \text{if } \tau \text{ is covered by } \Omega \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Then $\Omega$ is complete for $\mathcal{T}$ if.

$$\prod_{\tau \in \mathcal{T}} \text{Cover}(\Omega, \tau) = 1 \quad \Longleftrightarrow \quad \forall \tau \in \mathcal{T}, \text{Cover}(\Omega, \tau) = 1 \tag{13}$$

If transition $\tau \in \mathcal{T}$ is not covered by the current operator set $\Omega$, we need to account for that transition to ensure *Task Completeness*. Then, one or more additional operators $\{\mathcal{O}_{new}\}$ must be introduced, and the operator set $\Omega$ must be updated as: $\Omega \leftarrow \Omega \cup \{\mathcal{O}_{new}\}$.

This definition is enforced at the high-level system. In our experiments, condition (13) is guaranteed by ensuring that the final goal $f_{goal}$ of task $\mathcal{T}$ is captured by at least one operator's effect.

Moreover, the introduction of new operators may necessitate the incorporation of new fluents into our symbolic planner. These additional fluents extend the expressive power of the operator set by capturing semantic aspects that were previously unmodeled. Importantly, the newly introduced fluents are designed to correlate directly with waypoints in the continuous state space, thereby establishing a concrete linkage between the symbolic planning and the underlying state transitions.

In our hybrid HRL method, the standard operator set is {`open, close, reach, lift, move`}, which generally suffices for simpler manipulation tasks such as picking something. However, in more complex tasks, such as the cube-stacking scenario, the operator `move` is not sufficient to execute that task. This operator only places the end-effector with an object at an arbitrary position in the workspace without imposing the relational requirement needed for stacking. Hence, we introduce a task-specific operator called `stack` that defines waypoints to lift the end-effector *above* the target cube and release it. Therefore, we need a new fluent (*above*) to correctly learn the stacking sequence. Hence, the set $\Omega$ = {`open, close, reach, lift, move, stack`} is complete for cube stacking $\mathcal{T}$.

In a different task, involving door opening, the existing fluent set lacked a predicate capable of characterizing the door's open state, which, in this context, is defined by a rotation exceeding a certain angular threshold. To address this limitation, we introduced a new fluent, *rotated*, which explicitly captures the door's angular displacement, and correspondingly defined a new operator, `push`. Consequently, as we expand our method to new tasks and objects, we can systematically update the operator and fluent sets rather than redesign our entire planner structure to address new tasks.

### C. Hierachical Reinforcement Learning Based on Operators

In this work, we introduce a new method, inspired by SAC-X [5], to learn the agent's hierarchy by integrating the concepts of planning operators as part of the MDP definitions. The operator set $\mathcal{O}$ is the result of the decomposition of a long-horizon task, as explained in Section III-B. In contrast with the standard MDP, these new Operator-based MDPs (OpMDP), in

TABLE II
TRUTH TABLE OF THE OPERATORS' PRECONDITIONS ($\phi$, SHOW IN BLUE) AND EFFECTS ($\phi$, SHOW IN ORANGE)

| | open | | close | | reach | | lift | | move | | stack | | insert | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Fluent** | $\phi$ | $\psi$ | $\phi$ | $\psi$ | $\phi$ | $\psi$ | $\phi$ | $\psi$ | $\phi$ | $\psi$ | $\phi$ | $\psi$ | $\phi$ | $\psi$ |
| *openGripper* | F | T | T | F | T | – | F | F | F | F | F | T | F | T |
| *graspable* | F | – | T | – | F | T | T | – | T | – | T | – | T | – |
| *onTable* | T | – | T | – | T | T | T | F | F | – | – | F | – | F |
| *lifted* | – | – | – | – | – | – | F | T | T | T | T | – | T | – |
| *moving* | – | – | – | – | – | – | – | – | – | T | T | – | – | – |
| *above* | T | – | – | – | – | – | – | – | F | – | T | – | T | – |
| *onTop* | – | – | – | – | – | – | – | – | – | – | – | T | – | – |
| *inSlot* | – | – | – | – | – | – | – | – | – | – | – | – | F | T |
| *rotated* | – | – | – | – | – | – | – | – | – | – | – | – | – | – |

Fluents within the same column follow a logic `and`, except for fluents in red colour which follow a logic `or`.

eq. 14, include the set of operators $\mathcal{O}$ and their associated low-level policies, instead of the action set $\mathcal{A}$ in eq. 1. Furthermore, the OpMDP extends the dimension of the reward function to include multiple operators. Our method uses planning operators to define our on-demand high-level decision-making system based on the interaction with the environment. Since the high-level system selects $\mathcal{O}$ dynamically, we do not impose a fixed scheduling period, as done in SAC-X [5]. The planner has the important role of maximizing the expected return (see eq. 15) of the obtained plan by selecting the sequence of low-level control policies at run-time, instead of manually providing them as originally proposed.

$$\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{P}, a, r, \gamma\} \tag{14}$$

$$\mathbb{E}_{\pi(a|s,\mathcal{O})}\left[\mathcal{R}_{\mathcal{O}}(\pi_{t:t_f})\right] = \mathbb{E}_{\pi(a|s,\mathcal{O})}\left[\sum_{t=0}^{t_f}\gamma^t \mathcal{R}(s_t, a_t)\right] \tag{15}$$

To learn the low-level policies, we used the Soft Actor-Critic (SAC) method [41] to maximize the task return. This method is a model-free off-policy algorithm that maximizes the entropy ($\mathcal{H}$) of the policy:

$$\mathcal{H}(X) = \mathbb{E}_{(x\sim p)}\left[-\log p(x)\right], \tag{16}$$

where $X$ indicates a stochastic variable with $p(\cdot)$ representing the variable probability density function. In RL, $\mathcal{H}(\pi(\cdot|s))$ measures the randomness of the policy $\pi$ under state $s$, and the regularized entropy in SAC is the second term in the objective function, eq. 17.

$$J(\pi_{\mathcal{O}}) = \mathbb{E}_{\pi_{\mathcal{O}}}\left[\sum_{t=0}^{\infty}\gamma^t(R_{\mathcal{O}}(s_t, a_t) + \alpha\mathcal{H}(\pi_{\mathcal{O}}(\cdot|s_t)))\right], \tag{17}$$

where $\alpha$ is a regularization coefficient that determines the importance of the entropy, also called temperature. The entropy regularization increases the degree of exploration of the RL. A larger value of $\alpha$ enhances exploratory behaviour early in the learning process, helping to prevent premature convergence to local optima. The soft Q-function is defined as:

$$Q_{\mathcal{O}}(s_t, a_t) = R_{\mathcal{O}}(s_t, a_t) + \mathbb{E}_{\pi_{\mathcal{O}}}\left[\sum_{t=0}^{\infty}\gamma^t(R_{\mathcal{O}}(s_{t+1}, a_{t+1}) + \alpha\mathcal{H}(\pi_{\mathcal{O}}(\cdot|s_{t+1})))\right]. \tag{18}$$

SAC models two action-value functions $Q$ and a policy function $\pi$. Based on the idea of Double DQN [42], a network with a smaller $Q$ value is selected to alleviate the problem of overestimation of the $Q$ value. The loss function is defined in eq. 19 and aims to maximize the joint policy objective, similar to [33]:

$$\mathcal{L}(\pi) = \sum_{\mathcal{O}\in\mathcal{O}_{all}} \mathbb{E}_{s\sim\mathcal{B}, a\sim\pi_{\mathcal{O}}(\cdot|s)}\left[Q_{\mathcal{O}}(s,a) - \alpha\log\pi_{\mathcal{O}}(a|s)\right], \tag{19}$$

where $\pi$ refers to the set of policies $\{\pi_{\mathcal{O}_1}, \pi_{\mathcal{O}_2}, \cdots, \pi_{\mathcal{O}_K}\}$, $\mathcal{B}$ is the buffer that contains all of the transitions from interactions with the environment, starting from an initial state $s_0 \sim p_0$, following the policy $\pi(\cdot|s)$. By training each policy on states sampled according to the state visitation distribution of each possible task, these policies can successfully complete their respective tasks, regardless of the state in which the system was left by the policy for the previous task [5]. Therefore, it is safe to combine these learned policies.

For policy evaluation, soft Q functions aim to minimize the joint Bellman residual:

$$\mathcal{L}(Q) = \sum_{\mathcal{O}\in\mathcal{O}_{all}} \mathbb{E}_{(s,a,s')\sim\mathcal{B}, a'\sim\pi_{\mathcal{O}}(\cdot|s')}\left[(Q_{\mathcal{O}}(s,a) - \delta_{\mathcal{O}})^2\right]$$
$$\delta_{\mathcal{O}} = \mathcal{R}_{\mathcal{O}}(s,a) + \gamma(Q_{\mathcal{O}}(s',a') - \alpha\log\pi_{\mathcal{O}}(a'|s')) \tag{20}$$

### D. Combining RL and Symbolic Planning

For complex tasks, often the RL algorithm struggles to learn the right action sequence that could achieve the goal [43]. It has been demonstrated that RL can effectively learn policies for simpler tasks [44]. We hypothesise that complex tasks should be split into simple and independent actions, which could be learned with RL methods. Then, a different approach should be used to chain the actions to achieve the desired task. In this paper, we propose a method that demonstrates the above hypothesis by combining RL with symbolic planning for a hierarchical RL approach as shown in Figure 2. In our method, the chaining of low-level actions is performed automatically based on changes in the environment. The high-level agent selects an operator by matching the current preconditions. After executing its corresponding low-level policy, the observed effects define new preconditions, which guide the selection of the next suitable operator.$\mathcal{O}_i$ is

defined further as $(\phi(\mathcal{O}_i), \psi(\mathcal{O}_i))$, where the precondition $\phi(\mathcal{O}_i)$ must be *True* to be able to apply the operator $\mathcal{O}_i$, and the effect $\psi(\mathcal{O}_i)$ is the set of corresponding effects that describe a new environment state after the successful execution of the operator $\mathcal{O}_i$. The fluents used in this work are listed in Table I.

Environmental information is represented in a three-dimensional Cartesian coordinate system, where $x$ represents the horizontal axis, $y$ represents the depth axis, and $z$ represents the vertical axis. $p$ is the position of an object or the gripper, represented as $g$. The gripper position is defined in the middle point between two fingers, denoted as $p_g$. $\alpha$ and $\beta$ are two variables that can be instantiated by the objects of interest in the environment. Furthermore, we define the contact between two objects as:

$$contact(\alpha,\beta) \iff \min_{u\in\mathcal{S}_\alpha, v\in\mathcal{S}_\beta} \|u-v\| \leqslant \epsilon, \text{ with } \lim_{\epsilon\to 0}, \quad (21)$$

where for each object $\gamma \in \{\alpha,\beta\}$, $\mathcal{S}_\gamma \subset \mathbb{R}^3$ denotes the set of points on its surface. Here $\epsilon > 0$ is a small tolerance value that, ideally, approaches zero, ensuring that the objects are considered to be in contact only when their surfaces are essentially touching.

*1) Details About Operators:* Table II shows the relationship between the fluent and the operators. Table II only shows the must-satisfied conditions for each operator, where a dash ("–") indicates a trivial fluent value. For example, consider the operator `open` from Table II.

`open`:

$[1ex]\phi$ : $(\neg openGripper \wedge \neg graspable \wedge \text{onTable}) \vee above$

$[1ex]\psi$ : $openGripper$

This definition implies that the operator `open` can be activated only when its preconditions are satisfied. In particular, at the current time step, the gripper must be closed (i.e., $\neg openGripper$), the object must not be graspable (indicating that it is not in contact with the gripper), and the object must be on the table. We include the disjunctive condition $\vee$ to accommodate scenarios encountered in long-horizon tasks, such as STACK and INSERT, where the object is positioned above the target location; in these cases, the gripper should be opened to release the object regardless of the values of the other fluents. Upon successful execution, the fluent *openGripper* transitions from $F \to T$ as the effect of the operator `open`, while the values of the other fluents do not affect the main outcome under consideration.

Based on the operator definitions, some of the fluents are mutually exclusive. For example, in the operator `lift`, the fluents *onTable* and *lifted* are mutually exclusive.

*2) Learning Method and Evaluation:* The proposed learning algorithm is shown in Algorithm 1. HL and LL represent high-level and low-level respectively. Where the input variable $t_{traj}$ has two meanings: i) the maximum number of total trajectory time steps to terminate the RL episode, if the goal is not achieved; ii) time steps used to successfully achieve the goal within the maximum total steps. In Algorithm 1, `GetTargets` (line 9) identifies potential targets based on whether the objects are in contact with the table. Our method

**Algorithm 1** Hierarchical Training and Execution

**Input :** Planner check interval $\xi$, replay batch size $N$, total timesteps $t_{\text{traj}}$

1 Initialize replay buffer $\mathcal{B}$;
2 $t \leftarrow 0$, observe initial state $s \leftarrow s_0$, compute fluents $f \leftarrow L(s)$;
3 **while** $t < t_{traj}$ **do**
  `// HL: select next operator`
4  $\mathcal{O}_t \leftarrow$ `GetOperator`$(f)$;
5  $\tau \leftarrow 0$, success $\leftarrow$ false;
  `// LL: execute up to` $\tau < \xi$ `steps`
6  **while** $\tau < \xi$ ***and*** $\neg$ *success* **do**
7  $\tau \leftarrow \tau + 1$;
8  $a_t \sim \pi(a_0|s_t, \mathcal{O}_t)$;
9  *Execute* $a_t$, *observe next state* $s'$;
10  **if** $\tau \bmod \xi = 0$ **then**
11  $f \leftarrow L(s')$ ; `// HL: recompute fluents`
12  **end**
13  **if** $\psi(\mathcal{O}_t) \subseteq f$ **then**
14  success $\leftarrow$ true
15  **end**
16  *Store transition* $\langle s_t, a_t, s'_t \rangle$ *in* $\mathcal{B}$;
17  *Sample* $\{(s_i, a_i)\}_{i=1}^{N} \sim \mathcal{B}$;
18  **for** $\mathcal{O} \leftarrow 0$ **to** $K$ **do**
19  *Compute reward* $\vec{r}(s_t, a_t)$
20  **end**
21  *Update* $\pi$ *and* $Q$ *following eq. 19 and eq. 20*;
22  $t = t + 1$
23  $s \leftarrow s'$;
24  **end**
  `// HL: Prepare fluents for next operator`
25  **if** *success* **then**
26  $f \leftarrow \psi(\mathcal{O}_t)$;
27  **else**
28  $f \leftarrow L(s)$;
29  **end**
30  $t \leftarrow t + 1$;
31 **end**

is adapted from the RL sandbox,[4] which is a framework for RL algorithms. Optimization of the low-level policy in section III-C is done through reparameterization [45]. To mitigate overestimation bias in the Q-value estimates, the clipped double Q-learning algorithm [46] is used during training.

When evaluating a single operator's performance, the procedure differs from the training phase because no planning is involved. Instead, we manually specify the operator ID and compute its success rate. More details are shown in Algorithm 2.

*3) Reward and Success Criteria:* This section introduces the reward function and the success criteria used in our

[4]B. Chan, "Rl sandbox," https://github.com/chanb/rl_sandbox_public, 2020

**Algorithm 2** Evaluation for Single Operators

**input :** Planner period $\xi$, sample batch size $N$, max. evaluation time steps for all operators $t_{tot}$

*for each operator* $\mathcal{O}_1, \cdots, \mathcal{O}_i, \cdots, \mathcal{O}_K$, *subscript* $i$ *is operator id*
**while** $t < t_{tot}$ **do**
  *Interact with environment*
  $i \leftarrow$ `findFirstIndexWhere`$(count < t_{tot})$
  $count \leftarrow count + 1$
  *match* $i$ *with operator, get* $\mathcal{O}_i$
  $\pi_{\mathcal{O}_i} \leftarrow \mathcal{O}_i$
  $a_t \sim \pi(a_0|s_t, \mathcal{O}_i)$
  *Execute* $a_t$ *and observe next envState* $s'_t$
  *Assess whether* $\mathcal{O}_i$ *is successfully executed*
  **for** $\mathcal{O} \leftarrow 0$ **to** $K$ **do**
    *Compute reward* $\vec{r}(s_t, a_t, \mathcal{O}_i)$
  **end**
**end**

TABLE III
REWARD AND SUCCESS CRITERIA FOR EACH OPERATOR

| | Reward function elements | Success criteria |
|---|---|---|
| **open** | 1, 2, 3 | $a_g < 0$ |
| **close** | 1, 2, 3 | $a_g > 0$ |
| **reach** | 4 | distance between object and end-effect $< 0.02m$ |
| **lift** | 1, 3, 4, 5 | cube height $> 0.04m$ |
| **move** | 3, 4, 6, 7, 8 | cube vel. $> 0.05m/s$ acc. $< 5m/s^2$ |
| **stack** | 1, 3, 4, 6, 7, 9 | 1. all cubes contact<br>2. height between 2 contact cubes $> 0.035m$<br>3. only one cube contact with tray |
| **insert** | 3, 4, 9 | distance between cube and slot $< 0.003m$ |
| **push** | 4, 9 | $\theta_{\text{door}} > 20°$ |

method. Here, $a_g \in \mathbb{R}$ refers to gripper action, and $a \in \mathbb{R}^3$ is the distance from the target position to the end-effector. Generally, reward functions contain sparse and dense rewards because dense rewards could increase the speed of learning [33]. The following *Reward function elements* are used to formulate different operators' reward functions:

1) binary check for gripper action $a_g < 0$ means open otherwise means close
2) shaping term on Euclidean norm of $a$
3) binary reward if the distance between the end-effect and the target object $< 0.1m$
4) dense reward based on the distance between the end-effector and a target object, given by a linear or hyperbolic tangent function, defined heuristically
5) dense object height reward, given by a linear function
6) binary reward if the object reaches a certain height
7) object velocity reward, given by a linear function
8) object acceleration penalty, given by a linear function
9) dense reward based on the distance between the target position and the object's current position, given by a linear or hyperbolic tangent function, defined heuristically

The detailed reward function structures for each operator are listed in Table III.

*4) Training Procedure:* In our hierarchical framework, both high-level and low-level policies draw upon the same set of fluents as their reward basis in general, yet they differ in how these fluents are instantiated. At the high level, each fluent is treated as a binary predicate, evaluated via first-order logic formulations as detailed in Table II, for instance, the successful execution of the low-level policy `reach` changes the value of the fluent *graspable* to true. At the low level, each operator is equipped with an effect-driven reward function (Table III), such that the `reach` policy receives a reward proportional to the inverse distance between the gripper and the target. During the experiment, we didn't train a high-level policy; rather, we proposed an on-demand high-level system to define the next operator to execute, which calls the next low-level policy. The high-level decision-making system determines which high-level operator to execute by first grounding all fluents in the current environment state (as defined in Tables I and II). Each fluent is evaluated as either true or false based on the observations from the environment, and these truth values are then matched against the precondition set to define the next operator to execute at the low-level.

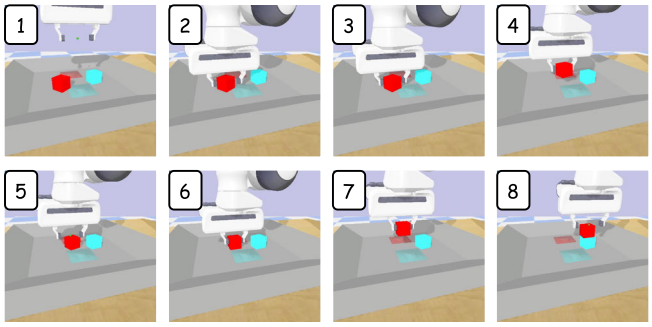


Fig. 6. Example of executing the learned sequence and policies for the task of stacking two cubes. At step 1, the agent `reaches` the red cube, then at step 2, since `reach` is finished, the next chosen operator would be `close` (step 3), then the operator `lift` is executed (step 4), but at step 5, an unexpected situation happens and the cube is dropped. Therefore, the agent infers that the next operator should be `reach` (6) and `lift` (7), finally the agent successfully `move` the cube and `stack` it on top of the blue cube (8), now the whole STACK task is finished. Note that this sequence was automatically determined by the agent.

## IV. EXPERIMENTAL VALIDATION

### A. Environment Setting

Our environment was adapted from the well-known manipulator-learning environment[5] (see Figure 6). Which contains a Franka Emika Panda manipulator, two cubes with dimensions $4cm \times 4cm \times 4$ $cm$, one tray ($30cm \times 30$ $cm$) with two slots ($4.1$ $cm \times 4.1$ $cm \times 1cm$), and sloped edges that keep the cubes within a reachable workspace. The range of motion of the end-effector, measured from the centre point of the tool, which is located directly between the gripper fingers, is confined within a $30cm \times 30$ $cm \times 30$ $cm$ box. The lower limit of this boundary is set to enable the gripper to engage with objects while avoiding collision with the bottom of the tray.

Actions related to movement involve three degrees of freedom (3)-DOF) for translational position shifts ($\Delta p$). This $\Delta p$

[5]T. Ablett, "manipulator-learning," https://github.com/utiasSTARS/manipulator-learning, 2022

is relative to the robotic arm's end-effector frame, and it is calculated by the difference between the target position, provided by the operator, and the current position of the end-effector. Then, this action is transformed into the robot's end-effector poses and mapped to joint positions. To achieve this, we use PyBullet's integrated function for position-based inverse kinematics, which generates the appropriate joint commands for the robotic arm.

In addition to these three dimensions, the action space includes a fourth dimension dedicated to the gripper's operation. This extra dimension is designed to be compatible with policy models that only produce continuous output values. Specifically, any positive real number commands the gripper to close, while any negative real number commands it to open.

The system operates at a rate of $20Hz$. For cube stacking and insertion tasks, each training episode is limited to 18 seconds, which is equivalent to 360 time steps for each episode. The positions of the cubes within the tray are randomized between episodes.

Another task analyzed in this work is the door-opening task. For this task, the door's position is fixed in each episode. Additionally, the end-effector's initial position is randomized to be located anywhere between $5\,cm$ and $14.5\,cm$ above the tray, adhering to previously defined bounds. Moreover, the gripper starts in a fully open state at the beginning of each episode. In this experiment, due to fewer operators, each training episode is limited to 13.5 seconds, which is equivalent to 270 time steps for each episode.

The operators we used in all our experiments are listed in Table II: `open`, `close`, `reach`, `lift`, `move`, `stack`, `insert`, and `push` with their own preconditions and effects. Note that we will use capital letters to indicate the name of the whole planning process, for example, "STACK" refers to stacking two cubes, which consists of different sequential operators, e.g., `reach`, `stack`, etc. (all labels in lowercase), i.e., STACK = {`reach`, `move`, ..., `stack`}. This symbolic structure of operators is a set of causal rules that outline the features of objects and environments, given by human experts.

During the training process, after the execution of one operator, there are two possible outcomes: success or failure. If successful, the next operator will be generated. In case of failure, the current operator will be executed for a maximum of 45 time steps. Afterwards, the environment state will be evaluated, and a new operator will be selected. This dynamic selection may trigger the same operator or a different one, depending on the measured environment.

### B. Experiment Results

We assessed the efficacy of our proposed approach through a dual-metric evaluation framework, focusing on both the chained policy execution success rate and the independent operator success rate.

*1) Chained Policy Execution Success Rate:* Chained policy execution for STACK and INSERT involves a sequence of operators. The construction of STACK is based on the specific set of operators: {`open`, `close`, `reach`, `lift`, `move`, `stack`}. We reuse the high-level structure to organize the order of operators for various tasks. For instance, in the INSERT task, we introduce an additional special operator, `insert`, to create holistic plans based on preconditions and effects, as presented in Table II. Thus, INSERT is constructed using the set of operators {`open`, `close`, `reach`, `lift`, `move`, `insert`}. By introducing task-specific operators and simple logical judgments in the high-level structure, it is possible to efficiently reuse, scale, and adapt to new tasks without having to redesign the entire system.

We test our method on three different seeds, and the chained policy execution success rate is presented in Figure 7. The experiment results achieved a near-perfect success rate of around 95% for the chained policy to stack two cubes after training for $1.5\,M$ environment interaction steps. Figure 6 represents a typical example that shows how our method uses chained policies for the STACK task. To ensure the planning process remains stable, we consider the execution of an operator to be successful only if it achieves its intended effect and this effect persists for at least $0.5\,s$.

INSERT is a task that requires a manipulator to insert one cube into a slot, depicted as a light-coloured area on the tray, as shown in Figure 8. The success rate of chained policy execution can reach approximately 85%.

*2) Individual Operator Success Rate:* Our method also learns independent low-level policies. Then, it is possible to evaluate individual operators, which can also be treated as a single policy. In figures 9 and 10, we show the performance comparisons between our method and 5 different baselines:

- *PFSM* [47]: A probabilistic finite-state machine (PFSM) employs conditional transition probabilities to determine and order the next high-level operator based on the current one.
- *QL* [48]: Use Q-learning to learn the high-level planner. Since in our method, operators are manually defined, this setting aims to learn a high-level plan automatically, as Q-learning is suitable for discrete variables.
- *SAC* [41]: Use flat SAC[6] as an ablation study.
- *BC* [2]: Though our method did not require human demonstrations directly. Symbolic representations and relations are knowledge that comes from humans, thus drawing an implicit relation with human experts. We further compare our method with the imitation learning baseline Behavioural Cloning (BC) [2], which learns a mapping from states to actions from human demonstrations (gathered from an RL expert [33]) explicitly.
- *Pretrained*: Pretrained a *move* model with operator set {`open`, `close`, `reach`, `lift`, `move`} for $1\,M$ time steps, achieved a success rate above 0.9 on `move`, then based on this model to train long-horizon tasks `stack` and `insert` with PFSM planner.

In the following part, we use the planner name PFSM and QL to represent the whole HRL model.

These comparisons demonstrate that our method not only improves training time and stability but also achieves higher

[6]We used the adapted version of [33] which made it compatible in a hierarchical structure, but with a uniform sampler to ablate the high-level decision-making system.

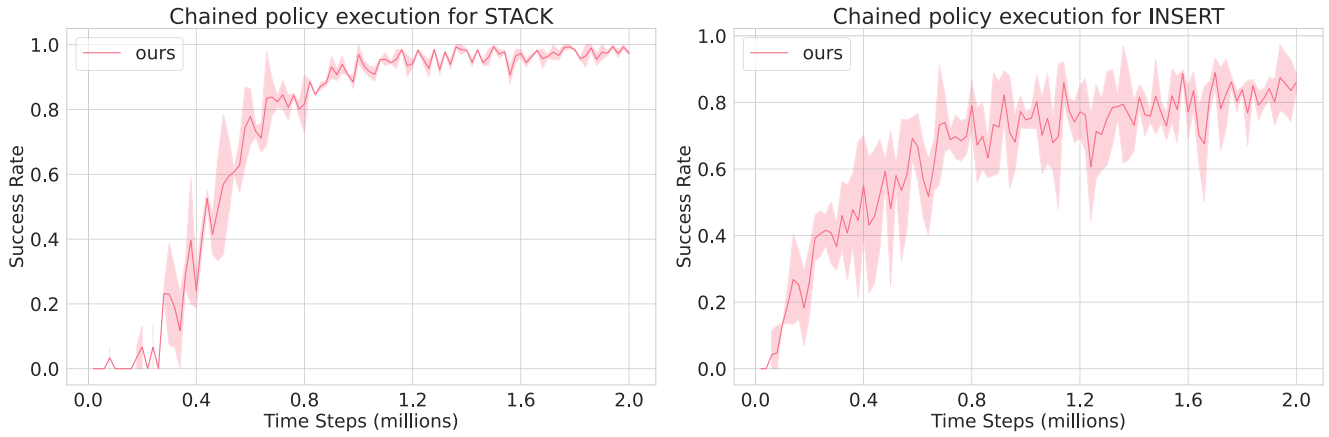


Fig. 7. Chained policy execution success rate for STACK 2 cubes and INSERT cube into a slot.

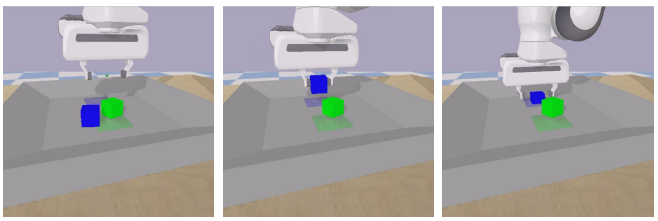

Fig. 8. Task for inserting the cube into the slot. From left to right, the images illustrate: (1) the initial randomized position, (2) the manipulator moves the blue cube to the target position, and (3) the manipulator inserts the blue cube in the light-coloured blue slot.

success rates on both standard single-operator tasks (e.g., `reach`, `lift`, `move`) and long-horizon single-operator tasks (e.g., `stack`, `insert`). We chose these specific algorithms because they represent the current state-of-the-art in hierarchical reinforcement learning, imitation learning, and transfer learning for robotic manipulation, thus providing a comprehensive benchmark to validate the efficacy of our approach.

For single low-level control policies like `reach`, `lift` and `move` in Figure 9, both our method and baselines learned the policies successfully, but our method is superior on training time and success rate, after training for 0.8 $M$ time steps, the success rate on average is: `reach` - 98.9%, `lift` - 99.7%, `move` - 97.4%. Our method shows a high task performance within a very short period and also a much more stable performance.

As for long-horizon manipulation, such as `stack` and `insert`, original SAC-X [5] needs 64.8 $M$ to learn (thus not included here), PFSM could not learn the tasks (see Figure 10), which further demonstrates our hypothesis discussed in section III-D.

Figure 6 (label 8) depicts the outcome of the operator `stack` in our method. The results in Figure 10 show that our method's success rate for stacking quickly surpassed 5 baselines, maintaining a high rate with a peak of 92% and an average of 85% after 0.8 $M$ training time steps. It is important to notice that even through transfer learning, the Pretrained model still needs 2.5 $M$ (including pre-training) to learn `stack` with a success rate over 80%. An additional experiment on the `insert` task also illustrates a better

TABLE IV
TRUTH TABLE OF OPERATOR `PUSH`

| Fluent | push $\phi$ | push $\psi$ |
|---|---|---|
| *openGripper* | – | – |
| *graspable* | T | – |
| *onTable* | – | – |
| *lifted* | – | – |
| *moving* | – | – |
| *above* | – | – |
| *onTop* | – | – |
| *inSlot* | – | – |
| *rotated* | F | T |

performance, and the success rate for our method would even increase if trained for a longer time. All experiments are tested over five different seeds.

*3) Scalability With a Small Sampling Experiment:* While our previous experiments (see Figures 7 to 10) demonstrated a competitive performance for complex long-horizon tasks such as stacking or insertion, within 2 $M$ environment interaction steps, we further evaluated our approach on a new door-opening scenario (Figure 11), considering smaller sampled experiment. This new experiment demonstrates the robust performance of our hybrid HRL method under substantially reduced sampling and with fewer operators, illustrating its ability to adapt across tasks of varying complexity. Specifically, we defined four essential operators for the door manipulation task (i.e., {`open, close, reach, push`}). It is worth noting that the standard operators `open, close, reach` were reused from previous scenarios, maintaining the same precondition and effect structure as in earlier experiments. We also introduce a new task-specific operator `push`, whose details can be found in Table IV, its success criterion is in Table. III.

In this new setting, we trained our hierarchical RL agent for only 0.1 $M$ time steps - a 95% reduction compared to our previous experiments. The door's position was initialized at a fixed location at the start of each episode. Despite this substantially smaller sampling, the agent successfully learned the strategy to chain individual operators to achieve

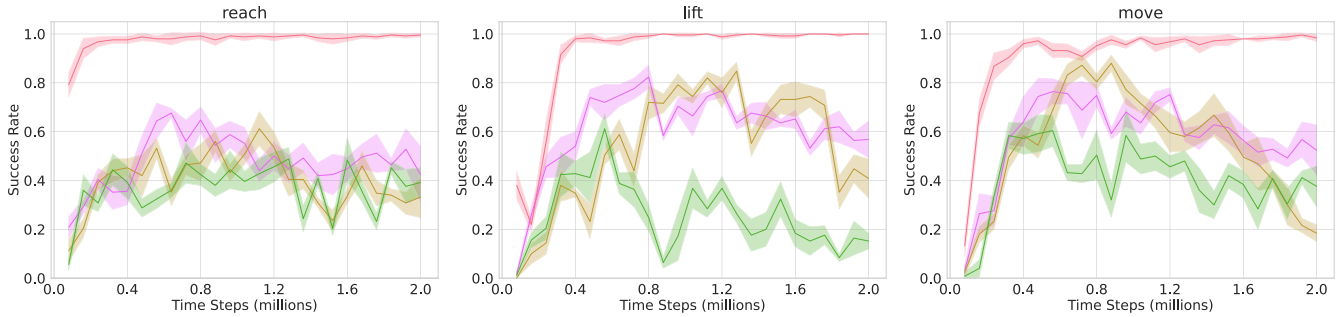


Fig. 9. Single operator success rate for standard tasks. Note that Figure 9 and Figure 10 share the same legend notation.

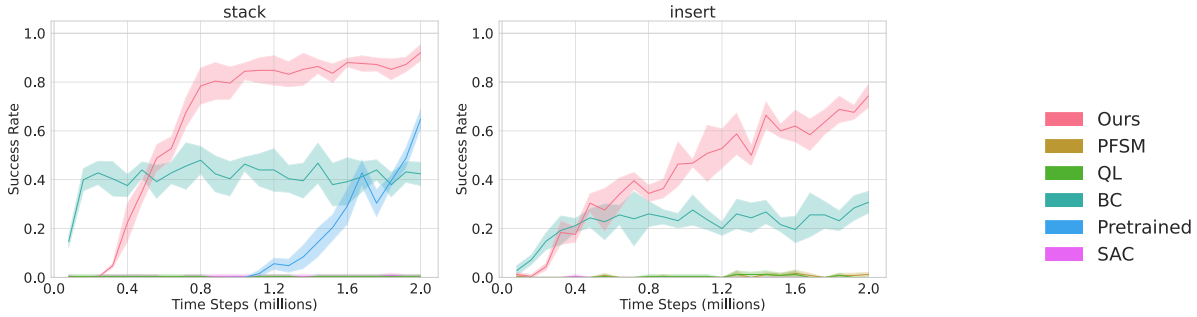


Fig. 10. Single operator success rate for long-horizon tasks.

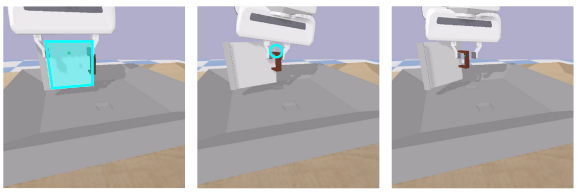

Fig. 11. Door-opening scenario. From left to right, the images illustrate: (1) the initial closed-door position, the highlighted area is the door; (2) the manipulator pushing the door, the grasping point is circled in the figure; and (3) the final state in which the door's angular displacement exceeds the specified threshold.

the long-horizon goal of opening the door. As shown in the left plot of Figure 12. The overall success rate reached approximately 84%.

As for individual operator evaluations, experiment results indicate that the newly introduced operator, `push`, also achieved an 85% success rate. Other operators (e.g., `reach`) similarly maintained a high success rate.

These results highlight the robustness and scalability of our proposed method: with a relatively small set of operators (four in this case) and reduced training steps, the agent still converges to a high success rate in both chained policies and individual operator executions. This is primarily attributed to the definition of operators, each operator only focuses on a narrowly defined goal and leverages symbolic preconditions to guide low-level policy selection. Consequently, even in tasks with fewer operators and simpler dynamics, our method maintains efficient exploration and fast convergence, reducing overall training demands without sacrificing performance.

*4) Operator Ablation:* To quantify the effect of the operator set size, we further performed an ablation study on the two-cube stacking task. Specifically, we compared a coarsened operator set, created by removing: (1) the `lift` operator and its associated *lifted* fluent; (2) the `move` operator and its *moving* fluent; (3) the `reach` operator, against the full, finer-grained operator set. Figure 13 illustrates the success rate of the individual `stack` operator when trained on operator sets with varying granularity. Although the ablated configuration involves fewer operators, the complete operator set converges to high success rates more rapidly, demonstrating superior learning efficiency. These findings corroborate those from our small-sample experiment in Section IV-B.3, demonstrating that the observed acceleration in convergence is not merely due to fewer operators but arises from the integration of symbolic abstractions with low-level policy learning, which yields denser reward feedback and improved sample efficiency.

*5) Dynamic Environment (Non-Static conditions):* Next, we rigorously assess the adaptability of our learned models (trained in static environments) in dynamic environments. We implemented four types of dynamic test scenarios to robustly evaluate our method under changing environmental conditions. Each test introduces a distinct dynamic behavior to examine the models' ability to adapt in real-time, recover from unexpected state changes, and maintain long-horizon task objectives under conditions not encountered during the static training.

We implemented four different types of dynamic test scenarios, which changed the behaviour on the target cube, base cube, or both, by continuously changing its velocity as follows:

1) *Random*: Application of continuous random linear velocity to one cube, $v_x, v_y \in [-0.2, 0.2]$
2) *Sinusoid*: Sinusoidal velocity applied to the designated cube (base, target, or both) along the x-axis with

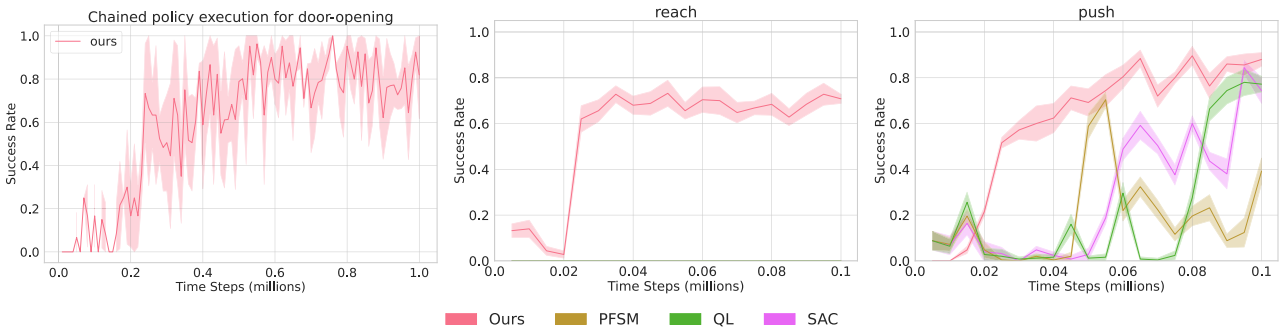


Fig. 12. Chained policy execution success rate for door-opening and single operator success rate.

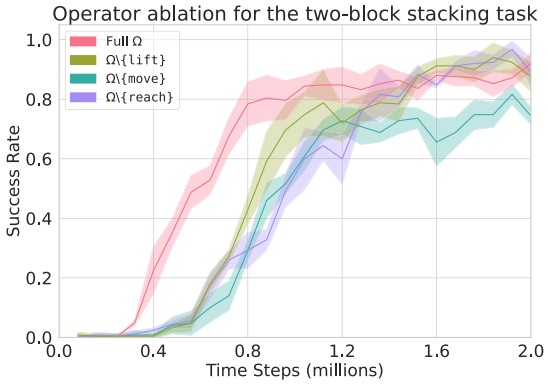


Fig. 13. Operator ablation studies. $\Omega$ = {`open`, `close`, `reach`, `lift`, `move`, `stack`}.

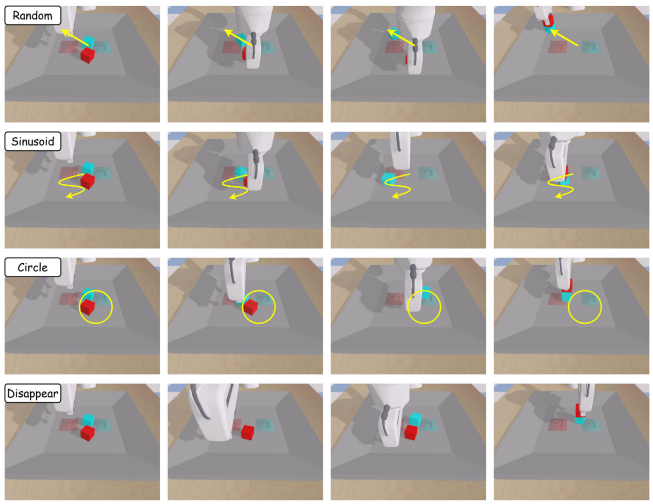


Fig. 14. Dynamic test scenarios for the base cube. From top to bottom, the images illustrate four different dynamic behaviours on the base cube: (1) random velocity, (2) sinusoidal trajectory, (3) circular trajectory, and (4) temporary disappearance. The agent has to handle these perturbations and successfully stack the target cube (red) on the base cube (cyan). Applying the dynamic behaviors to the target cube or both cubes produces similar effects.

linear velocity along the y-axis, generating a sinusoidal trajectory.

$$v_x(t) = 0.4\sin(2\pi t/T)$$
$$v_y(t) = 0.1 \tag{22}$$

$t$ denotes the current time step and $T$ is the period, here $T = 20\ s$.

3) *Circle*: Sinusoidal velocities applied to both $x$ and $y$ axes, generating a cube that moves on a circular trajectory. The trajectory is defined as:

$$x(t) = x_0 + 0.05\cos(2\pi t/T)$$
$$y(t) = y_0 + 0.05\sin(2\pi t/T) \tag{23}$$

here $T = 50\ s$, radius is 0.05, $x_0$ and $y_0$ is the randomized initial position of the cube.

4) *Disappear (discontinuous state)*: During execution, one of the cubes temporarily disappears for 20 time steps and then reappears in its original location. This condition simulates a momentary error in the robot's perception system.

We implemented the above types of dynamic test scenarios to robustly evaluate the models obtained with our method under different environmental conditions. These environments were not used during the learning phase. This means that the learned policies obtained from a static environment were tested in four different dynamic conditions to assess the adaptability of our method.

In these experiments, the presence of tray slots and collisions between cubes further exacerbate unpredictable situations, requiring the agent to adapt continuously in order to successfully complete the cube-stacking task. Figure 14 presents representative instances of the dynamic behaviors applied to the base cube. These behaviors can also be applied to the target cube, or both cubes simultaneously, resulting in a total of 12 environment settings. The agent's objective is to grasp the target cube (shown in red in Figure 14) and stack it on top of the base cube (cyan). Both cubes are randomly initialized within the tray area. Specifically, *disappear* is applied to the base cube at time step $t = 25$. This value was chosen empirically, as by this point, the robot has typically grasped the target cube and begun its approach toward the base cube. If the base cube disappears at this moment, the robot must detect the change and adapt its strategy accordingly. For the target cube, the *disappear* perturbation (discontinuous state) is introduced at time step $t = 5$, which corresponds to the moment just before the robot reaches and attempts to grasp it. When the perturbations are applied to both cubes, the base cube vanishes at $t = 65$, which is the empirical

TABLE V
SUCCESS RATES OF DIFFERENT METHODS UNDER DYNAMIC ENVIRONMENTS

| Method | Task | Static | Moved cube | Dynamic Environments | | | |
|---|---|---|---|---|---|---|---|
| | | | | Random | Sinusoid | Circle | Disappear |
| Pretrained | STACK | $0.860 \pm 0.032$ | base | $0.616 \pm 0.083$ | $0.540 \pm 0.095$ | $0.000 \pm 0.000$ | $0.620 \pm 0.020$ |
| | | | target | $0.844 \pm 0.105$ | $0.816 \pm 0.043$ | $0.816 \pm 0.033$ | $0.860 \pm 0.028$ |
| | | | both | $0.584 \pm 0.052$ | $0.496 \pm 0.075$ | $0.004 \pm 0.009$ | $0.568 \pm 0.058$ |
| | `stack` | $0.852 \pm 0.046$ | base | $0.588 \pm 0.050$ | $0.528 \pm 0.094$ | $0.000 \pm 0.000$ | $0.752 \pm 0.039$ |
| | | | target | $0.780 \pm 0.045$ | $0.712 \pm 0.059$ | $0.748 \pm 0.030$ | $0.804 \pm 0.067$ |
| | | | both | $0.580 \pm 0.063$ | $0.478 \pm 0.033$ | $0.004 \pm 0.009$ | $0.672 \pm 0.052$ |
| PDDLStream (linear) | STACK | $\mathbf{0.964 \pm 0.017}$ | base | $0.464 \pm 0.050$ | $0.160 \pm 0.047$ | $0.024 \pm 0.026$ | $0.056 \pm 0.022$ |
| | | | target | $0.388 \pm 0.050$ | $0.152 \pm 0.052$ | $0.000 \pm 0.000$ | $0.148 \pm 0.073$ |
| | | | both | $0.336 \pm 0.038$ | $0.060 \pm 0.035$ | $0.008 \pm 0.018$ | $0.132 \pm 0.023$ |
| | `stack` | N/A | | N/A | | | |
| PDDLStream (PRM) | STACK | $0.960 \pm 0.024$ | base | $0.428 \pm 0.097$ | $0.280 \pm 0.101$ | $0.052 \pm 0.023$ | $0.108 \pm 0.017$ |
| | | | target | $0.400 \pm 0.126$ | $0.124 \pm 0.038$ | $0.000 \pm 0.000$ | $0.280 \pm 0.051$ |
| | | | both | $0.304 \pm 0.052$ | $0.000 \pm 0.026$ | $0.004 \pm 0.009$ | $0.300 \pm 0.057$ |
| | `stack` | N/A | | N/A | | | |
| **Ours** | STACK | $0.936 \pm 0.033$ | base | $\mathbf{0.896 \pm 0.030}$ | $\mathbf{0.896 \pm 0.073}$ | $\mathbf{0.904 \pm 0.043}$ | $\mathbf{0.888 \pm 0.044}$ |
| | | | target | $\mathbf{0.888 \pm 0.044}$ | $\mathbf{0.884 \pm 0.026}$ | $\mathbf{0.912 \pm 0.030}$ | $\mathbf{0.848 \pm 0.061}$ |
| | | | both | $\mathbf{0.836 \pm 0.062}$ | $\mathbf{0.868 \pm 0.052}$ | $\mathbf{0.912 \pm 0.048}$ | $\mathbf{0.904 \pm 0.043}$ |
| | `stack` | $0.924 \pm 0.052$ | base | $\mathbf{0.904 \pm 0.043}$ | $\mathbf{0.960 \pm 0.020}$ | $\mathbf{0.952 \pm 0.030}$ | $\mathbf{0.848 \pm 0.048}$ |
| | | | target | $\mathbf{0.852 \pm 0.036}$ | $\mathbf{0.944 \pm 0.033}$ | $\mathbf{0.932 \pm 0.023}$ | $\mathbf{0.880 \pm 0.024}$ |
| | | | both | $\mathbf{0.808 \pm 0.076}$ | $\mathbf{0.912 \pm 0.064}$ | $\mathbf{0.956 \pm 0.026}$ | $\mathbf{0.836 \pm 0.017}$ |

**Task** STACK is achieved by the operators {open, close, reach, lift, move, stack}; "stack" is the independent policy.
*PDDLStream* is not applicable to independent policies such as "stack," since no predefined plan is available for execution.
Column **Static** lists the success rate in a static environment (for comparison).

interval after the target cube's disappearance, reappearance, and successful grasp. These dynamical conditions force the agent to generalize its low-level policy to unstructured, time-varying, and previously unseen conditions.

Since previous baselines (from Section IV-B.1 and IV-B.2) failed to achieve stacking, we did not apply these dynamical tests to them. Instead, for comparative purposes, we implemented three different baselines: (1) *Pretrained* model as RL baseline, for completeness, and given that our proposed system incorporates concepts from TAMP, we also implemented (2) the *PDDLStream (linear)* method [49], and (3) the *PDDLStream* (PRM) method [50] as representative TAMP baselines for a direct comparison in terms of action and motion plan generation. In both *PDDLStream* implementations, the domain is constructed to align with our formalization of fluents as predicates, and streams are employed to perform sampling-based motion planning. At the beginning of each operator execution, the trajectory generator retrieves the current target position, thereby allowing the system to adapt to dynamic environments where target states may vary due to external perturbations.

1) The *Pretrained* model was trained in the static cube-stacking environment for a total of $3M$ time steps, achieving an 85% average success rate, thus serving as a baseline.
2) In *PDDLStream (linear)*, we implemented a simple Cartesian trajectory generator with a linear velocity profile. The trajectory generator is defined as the following, where $\boldsymbol{p}_0$ and $\boldsymbol{p}_g$ are the initial and goal positions, respectively. The trajectory is discretized into $N$ steps.

$$\begin{aligned} \Delta \boldsymbol{p} &= \boldsymbol{p}_g - \boldsymbol{p}_0 \\ \boldsymbol{p}_k &= \boldsymbol{p}_0 + \frac{k}{N}\Delta \boldsymbol{p}, \quad k = 0, 1, \cdots, N \end{aligned} \tag{24}$$

3) In *PDDLStream (PRM)*, we utilize the Probabilistic Roadmap (PRM) [50] as the trajectory generator.

The results from these dynamical tests are detailed in Table V. For each situation, we ran 50 episodes over 5 random seeds, *Static* means no dynamic tests were applied. As shown in Table V, when the cubes remain static, our method and the baselines achieve a high success rate; the performance of our method is comparable to the best achieved by *PDDLStream (linear)* and *PDDLStream (PRM)*. However, it is important to note that both *PDDLStream* methods require an additional user-defined *stream model* to support motion planning and other geometric constraints, and that they generate plans offline. Consequently, the performance of these baseline methods is significantly affected by environmental dynamics and perturbations occurring at runtime. In contrast, our method makes decisions dynamically during operation, which allows it to maintain a high success rate with only minor or no degradation in performance. Moreover, ablation experiments conducted in the absence of a high-level system (i.e., only individual operator `stack`) remain at a stable high success rate under dynamic conditions. This finding suggests that incorporating an on-demand high-level decision-making system may enable the low-level policies to implicitly learn latent environmental variables from static environments that reflect the main operators' objectives. This, in turn, makes them inherently more robust to uncertainty and dynamic settings. Overall, these evaluations demonstrate the robustness of our method in handling dynamic environments.

## V. DISCUSSION

Performance comparisons between our approach and various baselines, depicted in the Figures. 9- 10, and shown in Table V, further underline the limitations of existing techniques in both long-horizon manipulation and adaptability to

dynamic environments. The stochastic nature of the underlying MDP often yields suboptimal state transitions and cumulative uncertainty, hindering RL from identifying optimal actions over long sequences. By contrast, our method dynamically selects the most appropriate policy via continuous integration of up-to-date environmental information and real-time feedback. As a result, it doesn't require an additional plan solver (e.g., PDDL) and could quickly adapt to dynamic environments, thereby enabling more informed and adaptive task execution. This underscores the critical importance of adaptive sequencing of low-level policies for achieving robust performance in dynamic and unpredictable settings.

The experimental results raise an intriguing question that will be considered in future work: *is there a direct correlation between the number of operators and the sample complexity of learning?* Another potential improvement is to capture more nuanced physical interactions, such as force or torque feedback. In the current door-opening setup, our agent relies on position-based actions. Incorporating force/torque sensing into the state space and reward function could greatly enhance robustness in tasks that require correct force control. Bridging symbolic operators with force/torque would thus be a natural next step, broadening the range of tasks that can be learned under our framework. Additionally, we will address the sim2real experiments in future work.

## VI. Conclusion

Our method tackles the challenge of long-horizon tasks in robotic manipulation by integrating symbolic planning with hierarchical reinforcement learning methods through operators. We introduced dual-purpose high-level operators, contributing both to generating holistic sequencing of actions with the possibility of executing them as independent policies. Experiments confirm a 97.2% success rate for the STACK chained policy execution and 85% for INSERT. Even an 85% average success rate is achieved for the individual operator `stack` in the absence of a holistic planning context. The small sampling experiment involved scaling down the operator sets by using fewer operators for the 'open the door' task. The results further highlight the robustness and scalability of our method, demonstrating a success rate of 84% for planning and executing the new long-horizon task and 85% for the individual operator `push`, all while reducing training time by up to 95%. We further tested four different dynamic settings, and our method holds a stable, high success rate of around 90% even without the need for re-training, surpassing different baselines. This indicates that high-level planning enhances the inherent robustness of low-level policies to disturbances. Our method thus offers a promising avenue for achieving both high task performance and efficient learning in the complex domain of long-horizon robotic manipulation.[7]

## Acknowledgment

The computations were carried out using resources provided by Chalmers e-Commons at Chalmers University of Technology.

[7]The code developed for this article is available with open access at: https://gitlab.com/craft_lab/rl-and-symbolic_planning/RL_operator

## References

[1] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.

[2] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," 2018, *arXiv:1805.01954*.

[3] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo, "A review of machine learning for automated planning," *Knowl. Eng. Rev.*, vol. 27, no. 4, pp. 433–467, Dec. 2012.

[4] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Amsterdam, The Netherlands: Elsevier, 2004.

[5] M. Riedmiller et al., "Learning by playing solving sparse reward tasks from scratch," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4344–4353.

[6] G. Prem, Y. P. J. Sin, V. Bytyqi, and E. Hayashi, "Enhanced deep reinforcement learning for robotic manipulation: Tackling dynamic weight in noodle grasping task," in *Proc. Int. Conf. Artif. Life Robot. (ICAROB)*, vol. 30, 2025, pp. 313–317.

[7] D. Tanneberg, E. Rueckert, and J. Peters, "Evolutionary training and abstraction yields algorithmic generalization of neural computers," *Nature Mach. Intell.*, vol. 2, no. 12, pp. 753–763, Nov. 2020.

[8] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty, "A review of learning planning action models," *Knowl. Eng. Rev.*, vol. 33, pp. 1–20, Sep. 2018.

[9] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, nos. 3–4, pp. 189–208, Dec. 1971.

[10] C. Aeronautiques et al., "Pddl—The planning domain definition language," Center Comput. Vis. Control (CVC), Yale Univ., New Haven, CT, USA, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998.

[11] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res.*, vol. 20, pp. 61–124, Dec. 2003.

[12] C. R. Garrett et al., "Integrated task and motion planning," *Annu. Rev. control*, vol. 4, no. 1, pp. 265–293, 2021.

[13] T. Xue, A. Razmjoo, and S. Calinon, "D-LGP: Dynamic logic-geometric program for reactive task and motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 14888–14894.

[14] T. Xue, A. Razmjoo, S. Shetty, and S. Calinon, "Logic-skill programming: An optimization-based approach to sequential skill planning," 2024, *arXiv:2405.04082*.

[15] T. Lin, C. Yue, Z. Liu, and X. Cao, "Modular multi-level replanning TAMP framework for dynamic environment," *IEEE Robot. Autom. Lett.*, vol. 9, no. 5, pp. 4234–4241, May 2024.

[16] J. Styrud, M. Mayr, E. Hellsten, V. Krueger, and C. Smith, "BeBOP—Combining reactive planning and Bayesian optimization to solve robotic manipulation tasks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 16459–16466.

[17] H. Wang, H. Zhang, L. Li, Z. Kan, and Y. Song, "Task-driven reinforcement learning with action primitives for long-horizon manipulation skills," *IEEE Trans. Cybern.*, vol. 54, no. 8, pp. 4513–4526, Aug. 2024.

[18] C. Voloshin, A. Verma, and Y. Yue, "Eventual discounting temporal logic counterfactual experience replay," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 35137–35150.

[19] C. Köprülü and U. Topcu, "Reward-machine-guided, self-paced reinforcement learning," in *Proc. Uncertainty Artif. Intell.*, 2023, pp. 1121–1131.

[20] H. Bourel, A. Jonsson, O.-A. Maillard, and M. S. Talebi, "Exploration in reward machines with low regret," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2023, pp. 4114–4146.

[21] J. Corazza, H. P. Aria, D. Neider, and Z. Xu, "Expediting reinforcement learning by incorporating temporal causal information," in *Proc. Causal Represent. Learn. Workshop NeurIPS*, 2023, pp. 1–10.

[22] W. Hatanaka, R. Yamashina, and T. Matsubara, "Reinforcement learning of action and query policies with LTL instructions under uncertain event detector," *IEEE Robot. Autom. Lett.*, vol. 8, no. 11, pp. 7010–7017, Nov. 2023.

[23] S. Han, M. Dastani, and S. Wang, "Sample efficient reinforcement learning by automatically learning to compose subtasks," 2024, *arXiv:2401.14226*.

[24] Y. Wen, S. Li, R. Zuo, L. Yuan, H. Mao, and P. Liu, "SkillTree: Explainable skill-based deep reinforcement learning for long-horizon control tasks," in *Proc. AAAI Conf. Artif. Intell.*, 2025, vol. 39, no. 20, pp. 21491–21500.

[25] W. Lu, M. Diehl, J. Sjöberg, and K. Ramirez-Amaro, "Leveraging symbolic models in reinforcement learning for multi-skill chaining," in *Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII)*, Jan. 2025, pp. 29–36.

[26] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–35, Jun. 2022.
[27] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," 2019, *arXiv:1910.11956*.
[28] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, "Multi-level discovery of deep options," 2017, *arXiv:1703.08294*.
[29] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," 2017, *arXiv:1712.00948*.
[30] Y. Jiang, S. Gu, K. Murphy, and C. Finn, "Language as an abstraction for hierarchical deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–15.
[31] J. Chen, T. Lan, and V. Aggarwal, "Hierarchical adversarial inverse reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 12, pp. 17549–17558, Dec. 2024.
[32] S. Sukhbaatar, E. Denton, A. Szlam, and R. Fergus, "Learning goal embeddings via self-play for hierarchical reinforcement learning," 2018, *arXiv:1811.09083*.
[33] T. Ablett, B. Chan, and J. Kelly, "Learning from guided play: Improving exploration for adversarial imitation learning with simple auxiliary tasks," *IEEE Robot. Autom. Lett.*, vol. 8, no. 3, pp. 1263–1270, Mar. 2023.
[34] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 30, 2020, pp. 540–550.
[35] J. Zhang, E. Dean, and K. Ramirez-Amaro, "Hierarchical reinforcement learning based on planning operators," in *Proc. IEEE 20th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2024, pp. 2006–2012.
[36] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 2970–2977.
[37] L. Guan, S. Sreedharan, and S. Kambhampati, "Leveraging approximate symbolic models for reinforcement learning via skill diversity," in *Proc. ICML*, 2022, pp. 7949–7967.
[38] S. A. Mehta and R. S. Zarrin, "On the feasibility of a mixed-method approach for solving long horizon task-oriented dexterous manipulation," in *Proc. IEEE-RAS 23rd Int. Conf. Humanoid Robots (Humanoids)*, Nov. 2024, pp. 949–956.
[39] H. Ma, T. V. Vo, and T.-Y. Leong, "Mixed-initiative Bayesian sub-goal optimization in hierarchical reinforcement learning," in *Proc. 23rd Int. Conf. Auto. Agents Multiagent Syst.*, 2024, pp. 1328–1336.
[40] A. M. Zanchettin, "Symbolic representation of what robots are taught in one demonstration," *Robot. Auto. Syst.*, vol. 166, Aug. 2023, Art. no. 104452.
[41] 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.
[42] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1, pp. 2094–2100.
[43] X. Yang et al., "Hierarchical reinforcement learning with universal policies for multistep robotic manipulation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 4727–4741, Sep. 2022.
[44] Q. Li, Y. Zhai, Y. Ma, and S. Levine, "Understanding the complexity gains of single-task RL with a curriculum," in *Proc. ICML*, 2022, pp. 20412–20451.
[45] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.
[46] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
[47] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines—Part I," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 7, pp. 1013–1025, Jul. 2005.
[48] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[49] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 30, 2020, pp. 440–448.
[50] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Apr. 1996.

**Jing Zhang** (Student Member, IEEE) received the B.Eng. degree in microelectronics science and engineering from the University of Electronic Science and Technology of China, China, in 2020, and the M.Sc. degree in systems, control, and mechatronics from the Chalmers University of Technology, Sweden, in 2024, where she is currently pursuing the Ph.D. degree with the Division of Systems and Control, Department of Electrical Engineering, also as a Ph.D. student with the WASP. Her research interests include robot learning, task reasoning and planning, and reinforcement learning.

**Emmanuel Dean** (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering (mechatronics) from CINVESTAV-IPN, Mexico City, in 2003 and 2006, respectively. From 2009 to 2020, he held a post-doctoral and a senior researcher positions at the Technical University of Munich (TUM), Munich, Germany. Since 2020, he has been a Senior Researcher with the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden. His research interests include embodied AI, physical human–robot interaction and collaboration, robust adaptive control of nonlinear robotic systems, user-centered robotic rehabilitation, and real-time multimodal sensor fusion.

**Karinne Ramirez-Amaro** (Member, IEEE) received the Ph.D. degree (summa cum laude) from the Department of Electrical and Computer Engineering, Technical University of Munich (TUM), Germany, in 2015. She has been an Associate Professor with the Department of Electrical Engineering, Chalmers University of Technology, since 2022. Previously, she was a Post-Doctoral Researcher with TUM. She is also a WASP-affiliated faculty. Her research interests include explainable AI, semantic representations, cause-based learning methods, collaborative robotics, and human activity recognition and understanding. She received several awards during her academic career, including the Laura Bassi Award, the price for an excellent doctoral degree for female engineering students, and the Google Anita Borg grant.