2025 IEEE/SICE International Symposium on System Integration (SII)
January 21-24, 2025, Munich, Germany

Leveraging Symbolic Models in Reinforcement Learning for Multi-skill Chaining*

Wenhao Lu[1], Maximilian Diehl[1], Jonas Sjöberg[1], Karinne Ramirez-Amaro[1]

***Abstract*— We envision robots learning new skills as efficiently as possible. A key challenge in this pursuit is that the efficiency of learning systems such as Reinforcement Learning (RL) deteriorates with the task complexity. For instance, in the task of building a tower of cubes, multiple subtasks must be executed in the correct sequence to succeed with this long-horizon task. As the sequence extends, determining the correct order of actions becomes increasingly difficult, particularly for RL methods that rely on trial-and-error. To tackle this, we propose a new method that integrates symbolic models into RL to boost learning efficiency in long-horizon tasks. Symbolic models offer task abstraction and can enhance sample efficiency for RL agents through high-level operators. Our approach focuses on task decomposition aligned with the structure of Automated Planning (AP) operators, enabling RL agents to synthesize individual skills for specific subtasks, thus they will require fewer learning samples. The task decomposition is designed with dual consideration: 1) reducing *errors* when linking subsequent skills together and 2) enhancing skill *reusability* for downstream tasks with similar structures. In simulated robot manipulation tasks, such as stacking two cubes, experiments demonstrate superior sample efficiency for our proposed approach (a 2x reduction in training cost) compared to most RL baselines. Furthermore, our method is robust to generalise to unseen rearrangement tasks with minimal interaction steps (fewer than 100), achieving an average success rate approximately 50% higher than baselines, which often struggle to make progress.**

I. Introduction

Robots are expected to learn new skills with maximal efficiency, minimizing the number of learning samples required. However, the efficiency of learning systems such as deep reinforcement learning (RL) [1]–[3] is closely tied to the complexity of tasks they encounter. For instance, training a robot to rearrange objects, like stacking a tower of cubes [4] requires executing a correct sequence of intermediate subtasks to complete the long-horizon task successfully. As the sequence grows longer, RL agents struggle to determine the optimal action sequence due to their reliance on trial-and-error learning, deteriorating their efficiency. Despite the advantages of RL in enabling robots to learn sophisticated and hard-to-engineer behaviours without additional task-level planning, this inefficiency persists. By revisiting the stacking example, it becomes clear that the skills required to stack one cube are identical to those needed to stack additional cubes (see Fig. 1). This observation suggests that breaking down long-horizon tasks into smaller, manageable portions for RL agents to solve individually can reduce the number of learning samples required. To achieve this goal, it demands a proper task decomposition.

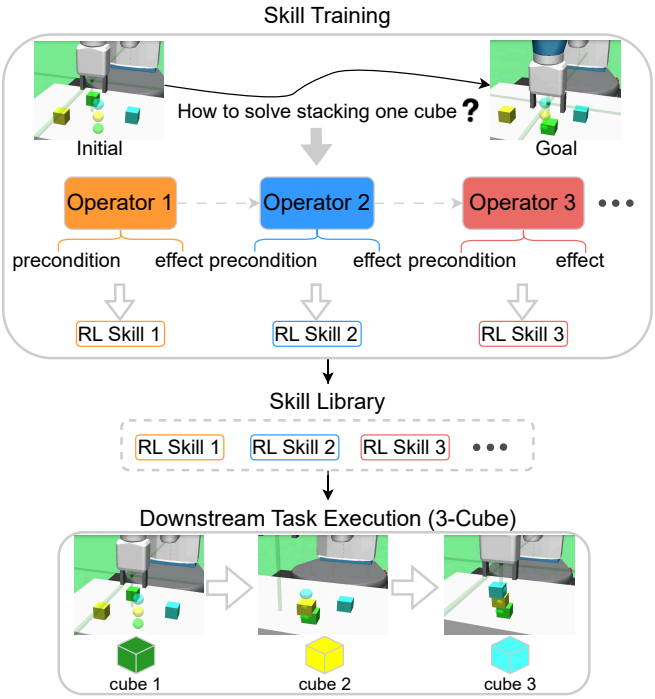


Fig. 1: Illustration of sequentially reusing and chaining RL skills, each learned independently for a single operator. These skills, originally developed for stacking *one* cube, are combined without additional learning to complete a *three-cube* stacking task.

Symbolic models, such as Automated Planning (AP) [5] allows for the rapid generation of action plans to tackle long-horizon tasks without the need for extensive trial-and-error interactions as required in RL. Yet, the prerequisite is to pre-define a set of possible high-level actions for the robots to sequence, and humans must explicitly detail the relevant symbolic knowledge for the given task. Further, when grounding high-level actions, AP heavily relies on strong assumptions about object properties, such as the known shapes of manipulated objects [6], which requires an additional execution layer to handle diverse objects.

This work focuses on coupling the advantages of both worlds to enhance the learning efficiency of RL agents: 1) we decompose a long-horizon RL task by utilizing the structure of AP operators, i.e., *preconditions* and *effects* (see Sec. III-B); 2) and we ground high-level actions using state-of-the-art RL techniques [2], [3]. Fig. 1 illustrates our approach, show-

*This work is supported by Chalmers AI Research Centre (CHAIR).
[1]Faculty of Electrical Engineering, Chalmers University of Technology, SE-412 96 Gothenburg, Sweden. {wenhaol, diehlm, jonas.sjoberg, karinne}@chalmers.se

ing how skills learned for subtasks decomposed by operators are reused to solve unseen tasks with similar structures. Our efforts are based on the insight that interactions between robots and objects can be characterized by a set of predicates describing relations and a set of events that cause changes in the environment, encapsulated within these predicates. These predicates are then used to define operators, which effectively decompose a long-horizon task.

The combination of operator structure and RL-based grounding allows for *reusability* [7], [8] of learned RL policies of each decomposed task, i.e., chaining them without finetuning for solving new tasks of similar structures, while an end-to-end RL policy fails, an observation in our experiments (Sec. V). For instance, a monolithic RL policy trained for stacking two cubes cannot stack a third cube. Yet, from the AP perspective, stacking 2 or 3 cubes involves the same operators —"reach a cube", "lift up a cube", and "stack on a cube". This reusability conserves computational resources, unlike end-to-end training, which often lacks sample efficiency.

Our method maintains a set of low-level policies/skills acquired for each subtask, as created by operators within the task's symbolic model. Concretely, we establish the initial and goal sets for a task based on preconditions and effects of its corresponding operator[1], and then train a goal-conditioned policy [9]. Notably, unlike previous approaches such as [10], [11], our learning process requires no expert demonstrations of tasks or the deliberate engineering of task rewards. The sequence of skills needed to compose a task specification is determined by action plans (Sec. III-B) generated by a PDDL-planner [12]. In summary, our contributions include:

1) A new method that fuses AP and RL for improved data-efficiency in long-horizon tasks, enabling RL agents to acquire and reuse skills for individual subtasks.
2) We instantiate our approach design in manipulation tasks of various types and demonstrate that it outperforms baselines on three long-horizon evaluation tasks (without extra learning), showcasing its superior generalisability through skill reusability.

II. Related Work

This work addresses the challenge of learning inefficiency in RL, intersecting multiple active research areas. The related work focuses on two key aspects: a) leveraging prior task knowledge, such as AP symbols, to enhance RL efficiency by reducing dependence on inefficient end-to-end learning, and b) accelerating RL by shifting the learning focus to higher-level abstractions when prior knowledge is provided as hard-coded action primitives or neural network-based skills.

Integrating Symbolic Planning into RL. To address the inefficiencies in RL for long-horizon tasks requiring extensive learning samples, previous work has integrated symbolic planning with RL. These approaches, such as those by [13], [14], used symbolic plans to guide RL agents for improving exploration efficiency and decision-making by enabling the learning of subpolicies for each subtask or transition within the generated plan. While effective for tabular problems, these methods do not address the complexities of robotic scenarios we focus on, which involve continuous and high-dimensional state-action spaces. [15] extended this integration to complex domains like Montezuma's Revenge (with raw pixel images as states and numerous object interactions), utilizing a planner–controller–meta-controller architecture to both generate optimal symbolic plans and corresponding subpolicies. However, this method required learning multiple subpolicies (e.g., 13 subtasks in Montezuma's Revenge [16]), many of which could be managed by a single, reusable policy (e.g., the policy for bringing the key to the left ladder could also be used for the right). Reusability of skills is crucial, as it reduces the required learning samples and enhances efficiency in long-horizon tasks. Our work builds on this by focusing on decomposing long-horizon tasks using an adequate symbolic model, emphasising the reusability of learned skills for downstream tasks with similar structures, thereby overcoming the limitations of earlier methods.

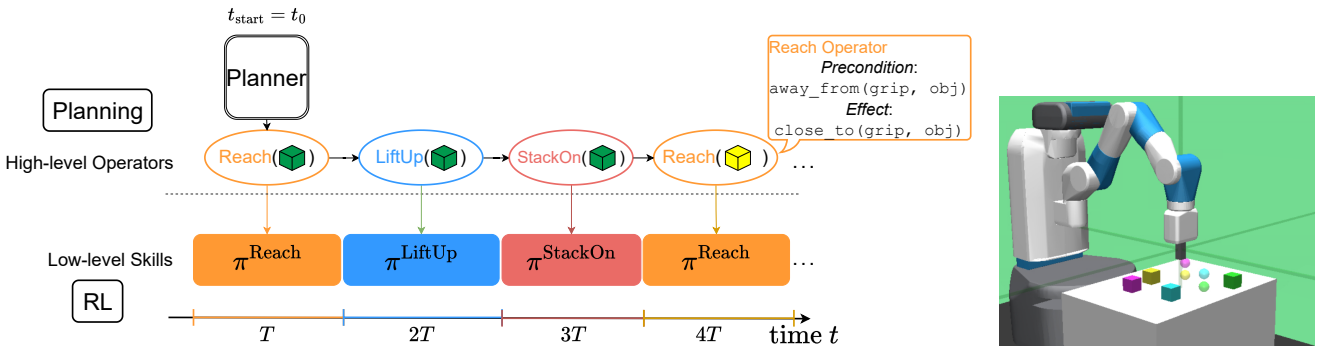
Recently, [17] combined RL with symbolic planning by learning individual RL skills for different planning operators. However, their approach is limited by the need for finely designed operators, which require numerous binary predicates that can be difficult to obtain, even for simple tasks like stacking one cube. In contrast, our approach achieves similar tasks with fewer operators, relying instead on readily available binary predicates, made possible by effective task decomposition (Sec. IV), which also ensures the reusability of learned skills, a feature not guaranteed by [17].

Furthermore, unlike previously mentioned works where learned skills are tied to fixed goal positions (e.g., a skill for lifting a cube cannot adapt to cubes positioned differently), our goal-conditioned skills [9] enable robust generalisation to varying goal positions within the same task, making them more sample-efficient.

Action Primitives/Skills Composition. MAPLE [1] is closely related to our work, as it uses a library of action primitives to solve long-horizon manipulation tasks. However, these primitives rely on hard-coded controllers, demanding manual effort and lacking generalisability as RL policies. Similarly, other approaches leverage RL-learned skills [18], [19] but require additional RL training to use these skills for downstream tasks, introducing challenges in efficiently sampling skills to guide downstream learning. In contrast, our approach enables the direct reuse of skills in a one-shot manner to solve downstream tasks with similar structures, without additional learning. Furthermore, while previous methods require handcrafted controllers or offline agent experiences (as sources of demonstration data) to extract skills, we utilise readily available operators (i.e., preconditions and effects) to guide skill learning, minimising the need for human intervention.

Gu et al. [8] similarly adopt the mechanism of decomposing a task into subtasks before re-composing the corresponding skills, but in a mobile manipulation context

[1]To enable this, we track predicates tied to operators, encoding environment changes in a structured form.



(a) Method Flow

(b) InTwoTower Task

Fig. 2: (a) Overview of our method using a task example involving (b) the manipulation of two towers of cubes (i.e., placing two cubes in each stack, with transparent balls indicating the goals). The robot sequentially performs high-level actions: reaching cubes (via Reach), lifting them (with LiftUp), and stacking them (using StackOn) into goal positions using its gripper. In the upper part (a), the planner generates high-level actions at $t = t_0$, while the lower part depicts the execution of low-level skills π that ground these actions. Each skill is executed within a fixed time budget T.

rather than our tabletop setup. Their decomposition is not systematic as it relies on the subtasks provided by the simulator. In contrast, our approach establishes a systematic decomposition using readily available planning operators, independent of external factors like the simulator used.

Lastly, our approach shares similarities with traditional top-down hierarchical reinforcement learning (HRL), which automates the decomposition of long-horizon tasks [20]. However, top-down HRL often suffers from instability at higher levels due to its reliance on environmental rewards [21], [22]. In contrast, our method uses a high-level planner to generate precise subtask plans without depending on low-level agent performance or environmental feedback. This avoids errors in sequencing skills for downstream tasks, made possible by our systematic task decomposition.

III. Preliminaries & Problem Formulation

A. Goal-conditioned Markov Decision Process

Standard reinforcement learning problems can be modelled by Markov Decision Process [23], where an MDP, represented as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r \rangle$, includes the state space $\mathcal{S} \subseteq \mathbb{R}^{d_{\mathcal{S}}}$, action space $\mathcal{A} \subseteq \mathbb{R}^{d_{\mathcal{A}}}$, transition probability function $p(s_{t+1}|s_t, a_t)$, and reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

In this work, we aim to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that takes the agent from any of the *initial set* $\rho_0 \subset \mathcal{S}$ to any of the set of states, denoted as the *goal set* $\mathcal{G} \subset \mathcal{S}$. To effectively address this problem, we extend the standard MDP by introducing a goal set $\mathcal{G}$, creating a goal-conditioned MDP as formally defined below, following [9].

Definition 1 (Goal-conditioned MDP)**.** Given an initial set ρ_0, and a goal set $\mathcal{G}$, a distance function d, a threshold $\epsilon \geq 0$, an MDP $\mathcal{M}$ with the following reward function

$$r_g(s, a) = \begin{cases} 0, & \text{if } \exists\, g \in \mathcal{G}, \text{ such that } d(s', g) \leq \epsilon \\ -1, & \text{otherwise} \end{cases}$$

where $s' \sim p(s'|s, a) > 0$ is the next state, is called a goal-conditioned MDP.

The introduction of a goal set into the MDP allows for the learning of an *universal* policy that can reach varying goals within a task, like reaching an object wherever it is put in a reach task. The general objective of a (goal-conditioned) MDP is to learn a policy that will maximise the expected cumulative reward

$$J(\pi) = \mathbb{E}_{s_t \sim p, a_t \sim \pi, g \sim \mathcal{G}}[\sum_{t=0}^{\infty} \gamma^t r_g(s_t, a_t)] \tag{1}$$

where γ is the discount factor satisfying $0 \leq \gamma < 1$.

Note that the non-negative reward (i.e., $r_g = 0$) depends on achieving goals. Thus, maximizing $J(\pi)$ will yield an optimal policy, called *skill*, that consistently attains specified goals from any initial state.

Alongside the goal-conditioned MDP and optimal policy, a task is defined by a tuple of an initial set ρ_0, a goal set $\mathcal{G}$ and a corresponding skill $\pi_{\mathcal{T}}$, denoted by $\mathcal{T} = \langle \pi_{\mathcal{T}}, \rho_0, \mathcal{G} \rangle$. Performing task $\mathcal{T}$ consists of following its skill $\pi_{\mathcal{T}}$ (to be learned) until reaching a goal state $s' \in \mathcal{G}$. Afterwards, another task can be selected to perform. Notably, $\pi_{\mathcal{T}}$ represents task-specific behaviours (i.e., the sequence of actions by following the skill), thus, the learned skill can be generalized to any object as long as its initial state and desired goal are within ρ_0, $\mathcal{G}$.

In many robotic tasks [24], learning a monolithic skill to achieve designated goals can be tough. In these scenarios, it is natural to split a full task $\mathcal{T}$ into a series of subtasks $\mathcal{T}_1, \mathcal{T}_2, \ldots$, each associated with a more manageable goal.

However, improper task splitting can lead to "hand-off" problems [25], where preceding (imperfect) skills terminate in states out of reach for succeeding ones, i.e., $\rho_0^{\mathcal{T}_2} \cap \mathcal{G}^{\mathcal{T}_1} = \emptyset$, with $\mathcal{G}^{\mathcal{T}_1}$ from task $\mathcal{T}_1$ and $\rho_0^{\mathcal{T}_2}$ from task $\mathcal{T}_2$. While prior works [7], [26] mainly focus on fine-tuning pre-learned skills to mitigate this issue, *we prioritize early task decomposition (before skill learning)* to enhance robustness against compounding errors–where early skill transition failures lead to unsuccessful chaining of later skills. Our task decomposition

design relies on planning operators from automated planning [5] .

B. Automated Planning

The goal of automated planning (AP) is to solve planning problems (e.g., robotic manipulation tasks) that are modelled by the so-called *planning domain* $\mathcal{P}$ [27], defined as $\mathcal{P} = \langle \mathcal{F}, A \rangle$, where $\mathcal{F}$ represents a collection of propositional symbols referred to as fluents in $\mathcal{P}$, which can either be true or false, and A encompasses the planning actions within the domain.

A fluent of arity of n is a predicate [2] with n object arguments, which describes relations over objects in the world environment [5]; a ground fluent is a fluent instantiated with particular objects. For a set of objects $\mathcal{U}$, planning states characterize situations in which the ground fluents (defined over those objects and termed $\mathcal{F}(\mathcal{U})$) are all true, denoted by $\mathcal{S}^{\mathcal{P}} \subset \mathcal{F}(\mathcal{U})$. Planning actions in $\mathcal{P}$ are provided by instantiating planning operators O, a higher-level abstraction of planning actions. For instance, a planning action for *grasping a cube A* is one instantiation of the operator `Grasp` with object *cube A* as its argument. Formally, a planning operator O is the blueprint of planning actions $\bar{a} \in A$ and holds a set of objects constituting its arguments, and a set of *preconditions* and *effects*, given as logical formulas over fluents. Preconditions govern what must be true for the operator to be applicable and effects describe how planning states $\mathcal{S}^{\mathcal{P}}$ would change after the operator is executed.

Since a planning action amounts to a grounded operator, we use the planning domain defined by $\mathcal{P} = \langle \mathcal{F}, O \rangle$ in the remainder of the article. A planning task is expressed by an initial planning state and a goal specification, given as a formula over ground fluents, as well as a plan, which is a sequence of (instantiated) operators that can be executed from the initial planning state and that leads to a goal.

IV. Planning Models in Low-level RL

We propose leveraging the operator structure (preconditions and effects) to effectively decompose tasks, implementing each operator using RL skills associated with the subtasks. The overall task is then solved by chaining these skills, guided by a high-level plan. The framework of our approach is illustrated in Fig. 2.

A. Key Aspects of Task Decomposition

Perfectly parsing a task into subtasks requires seamless initiation of each successive task upon the completion of the preceding one. This forms our initial proposition.

Proposition 1 (Composability)**.** Let $\pi_{\mathcal{T}_1}$ and $\pi_{\mathcal{T}_2}$ be skills of two consecutive subtasks $\mathcal{T}_1$, $\mathcal{T}_2$ respectively. If it holds $\rho_0^{\mathcal{T}_2} \supseteq \mathcal{G}^{\mathcal{T}_1}$, then the successful execution of $\pi_{\mathcal{T}_1}$ enables the execution of the next skill $\pi_{\mathcal{T}_2}$.

Following that, we notice the resemblance to the execution of planning operators in a plan, where the effect align with the precondition of the next. To communicate planning operators to MDP state space (on which a task is defined before), we begin by introducing the symbolic model, which, for a given (full) task $\mathcal{T} = \langle \pi_{\mathcal{T}}, \rho_0, \mathcal{G} \rangle$ and its corresponding planning domain $\mathcal{P} = \langle \mathcal{F}, O \rangle$, is specified as $\mathcal{E} = \langle \mathcal{P}, \chi \rangle$, where $\chi : O \to 2^{\mathcal{F}}$ is a function that outputs sets of fluents associated with planning operators.

The symbolic model allows us to identify fluents in the preconditions and effects of each operator, simplifying the construction of subtasks. We further assume a labelling function $L : \mathcal{S} \to 2^{\mathcal{F}(\mathcal{U})}$ to establish possible ground fluents (for a set of objects $\mathcal{U}$) [3] appearing in the domain whenever an MDP (low-level) state is reached; and the existence of a reversed labelling function $L^{-1} : 2^{\mathcal{F}} \to \mathcal{S}$. These labelling functions bridge between fluents and MDP states, promoting the task construction below.

B. From Planning Operators to Tasks

We employ the structure of operators to create subtasks, and these tasks, represented as $\mathcal{T}_1, \mathcal{T}_2, \ldots$, are constructed by defining their initial sets, ρ_0, and goal sets, $\mathcal{G}$, which correspond to the preconditions and effects of operators. Specifically, for every operator O, we build an initial condition and a termination condition. Each is defined as a set of low-level states in which all fluents involved in the preconditions or effects of operator O are observed. These conditions, in turn, serve as the initial set and goal set for a subtask $\mathcal{T}_O$ corresponding to operator O. Formally, we define the task decomposition using those conditions.

In the task decomposition, for every planning operator O, the corresponding task is defined by $\mathcal{T}_O = \langle \pi_{\mathcal{T}_O}, \rho_0^O, \mathcal{G}^O \rangle$, where ρ_0^O and $\mathcal{G}^O$ are both determined by the relation $L^{-1}(\chi(O)) \subset \mathcal{S}$. Here, L^{-1} translates the fluents of preconditions or effects of the operator O, represented by $\chi(O)$, back into the state space $\mathcal{S}$; and $\pi_{\mathcal{T}_O}$ is the skill trained to reach states in $\mathcal{G}^O$ from states in ρ_0^O. Then, a set of subtasks for the full task $\mathcal{T}$ can be defined as $\mathcal{T} = \{\langle \pi_{\mathcal{T}_{O_i}}, \rho_0^{O_i}, \mathcal{G}^{O_i} \rangle\}_{i=1}^{N}$, where N is the amount of operators.

C. Skill Learning

With the decomposed tasks $\mathcal{T} = \{\langle \pi_{\mathcal{T}_{O_i}}, \rho_0^{O_i}, \mathcal{G}^{O_i} \rangle\}_{i=1}^{N}$, we approach solving each task by using goal-conditioned RL algorithms [28] with Deep Deterministic Policy Gradient (DDPG) [2]. The core idea is to directly incorporate the goal $g \in \mathcal{G}^{O_i}$ into the policy $\pi(a|s, g)$, which is learned by maximising the objective defined in Eq. 1. This resulting goal-conditioned policy is behaviour-oriented and agnostic to object identities or dimensions, thus can be *generalised* to scenes featuring multiple cubes, e.g., the same skill can be applied twice to reach *cube A* and *cube B* appearing in one scene.

D. Skill Chaining

Now, each decomposed task possesses a distinct learned skill. These skills can be sequentially combined to accomplish any downstream task $\mathcal{T}$ during evaluation (achievable

[2]We use "predicate" and "fluent" interchangeably throughout this paper.

[3]In practice, L establishes ground fluents for low-level states of objects of interest.

using these acquired skills). A simple but effective way is to follow the order generated by a planner, such as Fast Downward [29], for task $\mathcal{T}$. The planner processes the problem description of the task in the *Plan Domain Definition Language* (PDDL) [30], which formalises predicates, operators, objects, the initial state, and the goal specification, all of which can be parsed into the planner. The planner then uses a heuristic function to generate a sequence of instantiated operators that achieve the goal. As the plan is executed, each operator maps to a specific learned skill, which is applied to the relevant objects defined by the operator's arguments.

V. Experiment

We conduct simulated robotic experiments to compare our skill chaining method with other baselines (see Sec. V-C). The following hypotheses are examined:

1) Is our approach more sample-efficient than tabula rasa policies with either dense or binary rewards?
2) Can we get an adequate success rate for the training task by chaining individually learned skills?
3) How does our skill chaining mechanism perform in evaluation tasks compared to other baselines in terms of generalization?

A. Environmental Setup

In experiments, we consider pick-and-place tasks of stacking cubes ($\# \geq 2$) in tower or pyramid configurations. That is, a robot end-effector is tasked to manipulate a number of cubes via a sequence of skills including *reach*, *grasp* and *stack*, to place them into desired 3D goal positions.

As there is a lack of environments allowing for individual learning of different skills in this pick-and-place setup, we create a set of new RL task environments on the MuJoCo physics simulator [31] which feature Gym-like API inspired by Fetch environments from OpenAI [16], [32].

States. The state space is comprised of both the state of *agent* and that of *surrounding objects* on a tabletop. An agent state includes proprioceptive quantities like end-effector's position ($\mathbb{R}^3$), linear velocity ($\mathbb{R}^3$), grip's joint angles ($\mathbb{R}^2$), and linear velocity ($\mathbb{R}^2$). For each object (cube), its state variable is described with positions, rotations, linear velocities, and rotational velocities ($\mathbb{R}^{15n}$, where n is the number of cubes involved in a task), i.e., $d_{\mathcal{S}} = 10 + 15n$.

Actions. A 4-dim vector, describing the relative displacement of end effector ($\mathbb{R}^3$) and the grip's opening state ($\mathbb{R}$), serves as the action space $\mathbb{R}^{d_{\mathcal{A}}}$ where $d_{\mathcal{A}} = 4$.

B. Tasks in Training and Evaluation

For all methods, we consider a training task of stacking two randomly spawned cubes in a tower, denoted by **InTowerOfTwo**. This kind of task's corresponding planning domain contains three operators: Reach, LiftUp, and StackOn. The set of fluents associated with those operators is reported in Table I and their grounding is depicted in Table II. Thus, three tasks are constructed from those operators and fluents: *reach a cube*, *lift up a cube*, and *stack on a cube*. Each is solved by learning a goal-conditioned DDPG policy [2], given its known initial set and goal set.

TABLE I: Operators with associated preconditions and effects for pick-and-place tasks.

Operator	Precondition	Effect
Reach	`away_from(grip, obj)`	`close_to(grip, obj)`
LiftUp	`close_to(grip, obj)`	`in_air(obj)` `grasped(obj)`
StackOn	`in_air(obj)` `grasped(obj)`	`in_goal(obj)` `released(grip)`

TABLE II: Symbolic fluents and their grounding in pick-and-place tasks.

Symbolic Fluents	Grounding
`away_from(grip,obj)`	gripper is 0.15 m away from `obj`
`close_to(grip,obj)`	gripper is within 0.075 m of `obj`
`in_air(obj)`	`obj` is at least 0.05 m off the tabletop
`grasped(obj)`	`obj` and gripper's two fingers are in contact
`released(grip)`	gripper is not in contact with any object
`in_goal(obj)`	`obj` is within 0.05 m of its goal position

Evaluation tasks maintain a similar structure to the training setup, with variations in features: *shape* (stack in tower or pyramid), $\#$*object*, and $\#$*tower*.

- **InTowerOfThree.** It extends the 2-cube training task, involving the stacking of three cubes into a single tower.
- **InTwoTower.** The goal is to stack two cubes into each of two separate towers (spaced apart).
- **InPyramid.** The aim is to sequentially stack 3 cubes into a pyramid of 2-1 with predefined ordering.

C. Baselines and Implementation

Our approach, labelled Skill in Chain (SIC), maintains a skill set, each linked to an operator and chainable as directed by a PDDL planner for downstream tasks.

One baseline, **Landmark-Rewarding (LR)**, trains a tabula rasa policy using a dense reward for the 2-cube training task, where rewards are based on the # landmarks achieved during learning. These landmarks, derived from operators' effects (e.g., `in_air(obj)`), act as subgoals. Unlike SIC, the LR policy focuses on maximising subgoal completion without regard to their relative ordering. For comparison with curriculum learning, we use the **Curriculum-Based (CB)** approach, training a monolithic policy across a predefined set of three task stages with increasing complexity–starting with placing a cube, then stacking one, and finally stacking two. Each task offers binary rewards, following the setup in [28]. Lastly, we use a tabula rasa policy to tackle the training task with sparse incremental rewards based on the correct placement of blocks (r = *no. of cubes in place* $-$ *no. of all cubes*), denoted as **Binary-Policy (BP)**. Unlike LR, BP does not incorporate task structure (landmarks) in its learning.

VI. Results & Evaluation

We evaluate our SIC approach against three baselines—LR, CB, and BP—based on the following criteria and their corresponding learning outcomes:

- *sample efficiency*: first time $t_{80\%}$ to an 80% success rate in stacking two cubes in **training**.
- *reusability*: success rate in completing **evaluation** tasks over 20 episodes, averaged across 5 seeds for each task environment.
- *progressive completion rate*: success rate of individual skills used within SIC for solving **evaluation** tasks.

For the last two metrics, evaluation is conducted on three tasks: InPyramid, InTowerOfThree, and InTwoTower. Notably, we do not report evaluation statistics for the monolithic approach BP, as it did not achieve progress in stacking cubes due to its sparse reward signal. In fact, we employ a planner to generate a task plan, and during execution, we substitute the planned operator with a skill.

TABLE III: Success rates for all approaches on both evaluation and training tasks over 20 episodes (averaged across 5 seeds). Discussion of the SIC w/o Reach is in Sec. VI-B.

Approaches	**Training task**	**Evaluation task**		
	InTowerOfTwo	InPyramid	InTowerOfThree	InTwoTower
SIC (our)	$90.0 \pm 0.0\%$	$68.0 \pm 10.2\%$	$64.0 \pm 12.8\%$	$54.0 \pm 8.0\%$
SIC w/o Reach (our)	$95.0 \pm 0.0\%$	$\mathbf{85.0} \pm 8.4\%$	$\mathbf{77.0} \pm 7.5\%$	$\mathbf{63.0} \pm 7.5\%$
Curriculum-Based	$\mathbf{100.0} \pm 0.0\%$	$65.0 \pm 7.7\%$	$0.0 \pm 0.0\%$	$1.6 \pm 1.4\%$
Landmark-Rewarding	$88.0 \pm 6.0\%$	$8.0 \pm 8.7\%$	$0.0 \pm 0.0\%$	$4.8 \pm 1.2\%$
Binary-Policy	$0.0 \pm 0.0\%$	$0.0 \pm 0.0\%$	$0.0 \pm 0.0\%$	$0.0 \pm 0.0\%$

A. Comparative Analysis

Sample Efficiency in Learning. To empirically demonstrate the sample complexity of all methods, we plot the evolution of their success rate during training in Fig. 3, along with the corresponding convergence speed (i.e., $t_{80\%}$), as depicted in Fig. 4. Our approach SIC and curriculum-based CB achieve a 100 % success rate at convergence, while LR converges to around 80% despite demanding a $2\times$ larger number of samples.

Moreover, Fig. 4 indicates that non-monolithic approaches (SIC and CB) are more sample-efficient than monolithic ones (LR) during training. Furthermore, learning in curriculum (from CB) is also comparable to learning individual skills as in SIC, in terms of $t_{80\%}$. However, as SIC is inherently designed to allow for more flexibility in reusing induced skills, it can be expected to show superior generalizability than others, as reflected in its highest success rates across evaluation tasks in Table III.

Reusability Performance. We compare the test performance of all methods in three evaluation tasks and the task they were trained on, and the results are summarized in Table III. It shows that method SIC significantly outperforms all the baselines in all evaluation tasks, though CB achieves a slightly better success rate in training. In terms of generalizability, our approach benefits from reusability and thus readily performs well in configurations with more cubes or different shapes, whilst other baselines struggle to squeeze into a passing line around 10%. Though, in task InPyramid, CB exhibited its decent ability to generalize, it fails in other configurations. This highlights the uniqueness of our approach design for reusing off-the-shelf skills, with potential for further extensions using additional skills (e.g., inserting and wiping objects) in the task design for a broader range of downstream tasks.

Progressive Completion Rate for SIC. In the evaluation phase, we sequentially apply the learned skills in SIC (using a PDDL-planner) to solve new tasks and measure whether the goal of a subtask (e.g., reaching one cube) is completed by utilizing its corresponding fluents (Table II). A subtask is considered accomplished when all fluents associated with its goal are satisfied. It is observed that the progressive completion rate monotonically decreases as only if the current one is completed we proceed to the subsequent subtask, as depicted in Fig. 5 (due to space constraint, we present the example of InTwoTower task). It's important to note that subtask *stack_i* merely requires placing the cube within its goal region, but the placement might be unstable as the cube can fall down on the table afterwards, potentially causing a failure in the full task, whose success rate is depicted in orange in Fig. 5. Hence, while our approach promises composability in design, there exist unpredictable errors in the execution of skills, which occasionally disrupt the composability. A detailed analysis of it is provided next.

B. Discussion

The set of operators, determined by the planning domain, guides skill learning in our SIC approach. To assess the impact of high-level modelling (i.e., defining the planning domain) on low-level execution (i.e., skill chaining), we tested a variant of SIC without the Reach skill. The results showed that the three-skill scheme performed slightly worse than the two-skill scheme (LiftUp and StackOn) in terms of full task completion during deployment (Table III). This aligns with findings in [1], which suggests that non-prehensile tasks (without contact-rich interactions, such as peg insertion) may not require the *reach* skill. In addition, introducing the reach skill can sometimes lead to unexpected chaining errors. For example, executing a reach skill may accidentally knock down a cube previously positioned by a stack skill. This compounding issue can result in a lower full-task success rate despite high subtask/skill completion rates (see Fig. 5). One potential solution (and future work) is to incorporate low-level skill feedback into the high-level planner to trigger re-planning or task restart upon failures, or reconfigure the planning domain by selecting more suitable operators based on RL feedback. Despite these issues, SIC with three skills still achieved adequate performance in all evaluation tasks compared to other baselines.

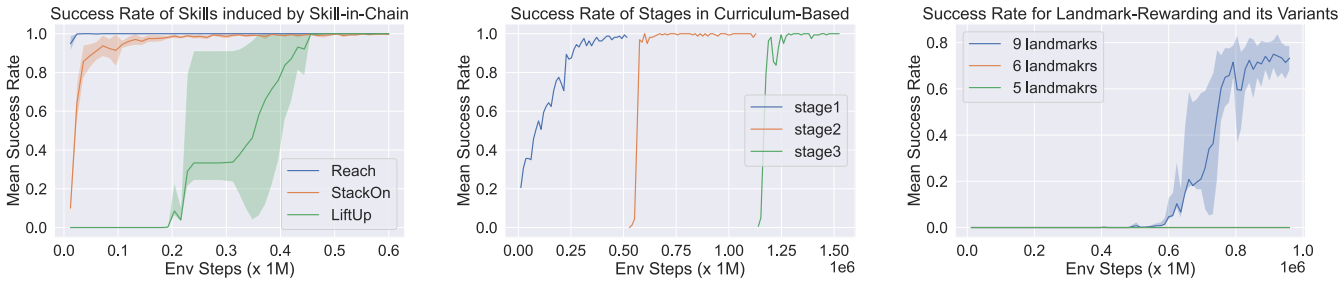


Fig. 3: Mean training success rates with standard deviations (over 4 seeds, except CB with one seed) for methods SIC (left), CB (middle), and LR (right). The x-axis indicates learning steps, and the y-axis represents success rates. SIC displays success rates for three concurrent skills, while CB shows success rates for three consecutive stages. LR's success relies on the number of landmarks being appropriately set.

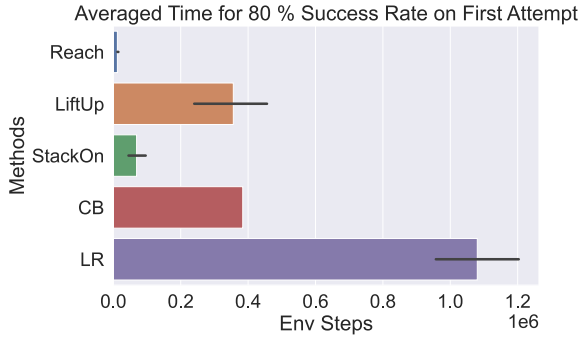


Fig. 4: Average $t_{80\%}$ steps to reach an 80% success rate in the training task. First four rows correspond to non-monolithic approaches (the first three for SIC), and the fifth row is the monolithic approach (LR).

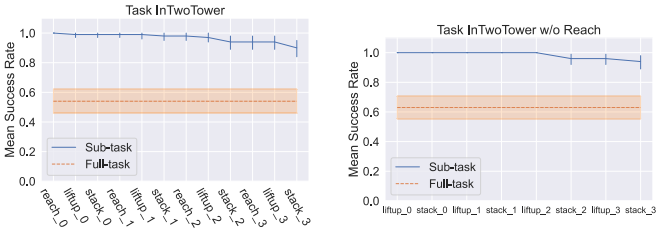


Fig. 5: Progressive completion rates (solid blue line with mean and one *std* deviation error) for SIC and SIC w/o Reach in the InTwoTower task. The x-axis displays progressive subtasks, while the y-axis indicates the completion rate for each subtask. The orange dotted line shows the success rate of the full task (mean with one *std* deviation).

VII. Conclusion

We conceive an approach for efficiently solving long-horizon tasks by decomposing them into subtasks using operators from their symbolic models. The experiments demonstrate a strong generalization of our approach to new, diverse pick-and-place tasks, outperforming most baselines due to the reusability of pre-trained skills, and enhancing sample efficiency for similar tasks. Skill chaining with minimal compounding errors is the key. While we apply this approach to pick-and-place tasks, it can be easily adapted to other robotic tasks, including prehensile and non-prehensile ones, by diversifying the skill pool. One implication of our approach is that, with a series of learned skills, we can readily explain agent failure behaviour during skill execution by identifying the stage at which a specific skill failed. For real-world applications, skills could be pre-learned in simulation with augmented noise using domain adaptation techniques, and then transferred to the real environment, provided symbolic fluents are effectively tracked to ensure reliable robotic operation. Exciting avenues for future work could involve integrating low-level skill feedback into high-level planning for re-planning after failures and automating task decomposition from expert demonstrations.

References

[1] S. Nasiriany, H. Liu, and Y. Zhu, "Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[4] K. Ehsani, W. Han, A. Herrasti, E. VanderBilt, L. Weihs, E. Kolve, A. Kembhavi, and R. Mottaghi, "Manipulathor: A framework for visual object manipulation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4497–4506.

[5] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, ser. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam: Morgan Kaufmann, 2004. [Online]. Available: http://www.sciencedirect.com/science/book/9781558608566

[6] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3684–3691.

[7] Y. Lee, J. J. Lim, A. Anandkumar, and Y. Zhu, "Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization," in *Conference on Robot Learning*. PMLR, 2022, pp. 406–416.

[8] J. Gu, D. S. Chaplot, H. Su, and J. Malik, "Multi-skill mobile manipulation for object rearrangement," in *The Eleventh International*

Conference on Learning Representations, 2023. [Online]. Available: https://openreview.net/forum?id=Z3IClM_bzvP

[9] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1312–1320. [Online]. Available: https://proceedings.mlr.press/v37/schaul15.html

[10] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4637–4643.

[11] M. Diehl, C. Paxton, and K. Ramirez-Amaro, "Automated generation of robotic planning domains from observations," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6732–6738.

[12] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," 08 1998.

[13] M. Grounds and D. Kudenko, "Combining reinforcement learning with symbolic planning," in *European Symposium on Adaptive Agents and Multi-Agent Systems*. Springer, 2005, pp. 75–86.

[14] F. Yang, D. Lyu, B. Liu, and S. Gustafson, "Peorl: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 4860–4866.

[15] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 2970–2977.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[17] J. Zhang and K. Ramirez-Amaro, "Hierarchical reinforcement learning based on planning operators," *arXiv preprint arXiv:2309.14237*, 2023.

[18] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Conference on robot learning*. PMLR, 2021, pp. 188–204.

[19] K. Rana, M. Xu, B. Tidd, M. Milford, and N. Sünderhauf, "Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics," in *Conference on Robot Learning*. PMLR, 2023, pp. 2095–2104.

[20] M. Hutsebaut-Buysse, K. Mets, and S. Latré, "Hierarchical reinforcement learning: A survey and open research challenges," *Machine Learning and Knowledge Extraction*, vol. 4, no. 1, pp. 172–221, 2022. [Online]. Available: https://www.mdpi.com/2504-4990/4/1/9

[21] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 3540–3549.

[22] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine, "Latent space policies for hierarchical reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1851–1860.

[23] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[24] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2021, pp. 726–747.

[25] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets, *et al.*, "Habitat 2.0: Training home assistants to rearrange their habitat," *Advances in neural information processing systems*, vol. 34, pp. 251–266, 2021.

[26] Y. Lee, S.-H. Sun, S. Somasundaram, E. S. Hu, and J. J. Lim, "Composing complex skills by learning transition policies," in *Proceedings of International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rygrBhC5tQ

[27] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105

[28] J. B. Lanier, S. McAleer, and P. Baldi, "Curiosity-driven multi-criteria hindsight experience replay," *CoRR*, vol. abs/1906.03710, 2019. [Online]. Available: http://arxiv.org/abs/1906.03710

[29] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, no. 1, pp. 191–246, 2006.

[30] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li, "Recent trends in task and motion planning for robotics: A survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[31] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[32] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *CoRR*, vol. abs/1802.09464, 2018. [Online]. Available: http://arxiv.org/abs/1802.09464