Continual Robot Learning via Language-Guided Skill Acquisition

Anonymous Author(s)

Abstract

To support daily human tasks, robots need to tackle complex, long-horizon tasks and continuously acquire new skills to handle new problems. Deep Reinforcement Learning (DRL) offers potential for learning fine-grained skills but relies heavily on human-defined rewards and faces challenges with long-horizon goals. Task and Motion Planning (TAMP) are adept at handling long-horizon tasks but often need tailored domain-specific skills, resulting in practical limitations and inefficiencies. To overcome these complementary limitations, we propose LG-SAIL (Language Models Guided Sequential, Adaptive, and Incremental Skill Learning), a framework that leverages Large Language Models (LLMs) to synergistically integrate TAMP and DRL for continuous skill learning in long-horizon tasks. Our framework achieves automatic task decomposition, operator creation, and dense reward generation for efficiently acquiring the desired skills. To facilitate new skill learning, our framework maintains a symbolic skill library and utilizes the existing model from semantic-related skills to warm start the training. LG-SAIL demonstrates superior performance compared to baselines across six challenging simulated task domains across two benchmarks. Furthermore, we demonstrate the ability to reuse learned skills to expedite learning in new task domains, and deploy the system on a physical robot platform. More results on website: https://sites.google.com/view/continuallearning.

CCS Concepts: • **Computing methodologies** → **Planning with abstraction and generalization**; **Robotic planning**; **Sequential decision making**; **Knowledge representation and reasoning**.

Keywords: LLMs for Robotics, Skill Abstraction and Planning, RL and Sequential Decision-Making

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Anonymous Author(s). 2018. Continual Robot Learning via Language-Guided Skill Acquisition. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

For robots to aid in daily human tasks, they need to tackle intricate long-term challenges and adapt to unfamiliar situations. While Deep Reinforcement Learning (DRL) techniques are promising for acquiring fine-grained manipulation skills [11, 19], they require carefully designed reward functions and often fail to achieve long-term goals in complex environments. Conversely, Task and Motion Planning (TAMP) methods [9] are adept at addressing and adapting to long-term tasks due to their robust state and action abstracts. However, their dependence on expertise to construct the planning domain restricts their practical use in real-world scenarios.

In an effort to overcome the constraints of both TAMP and skill learning, researchers have proposed the concept of learning skill policies within TAMP systems [4, 27, 28]. While these approaches have shown promise in enabling skill acquisition for long-horizon tasks efficiently, they still rely on expert knowledge to define planning domains (e.g., symbolic operators with conditions and effects) and dense reward functions. These assumptions limit the scalability of skill learning, especially in real-world contexts where robots frequently confront novel challenges where the planning domains cannot be defined beforehand.

In this work, we introduce LG-SAIL (Fig. 1), a continuous learning framework for robot manipulation that integrates large language models (LLMs) with TAMP and DRL. We observe that LLMs, trained on vast web data, excel at commonsense reasoning for task decomposition and skill creation, while TAMP's compact predicate-based state representations provide strong priors that guide and regularize LLM reasoning, improving efficiency and robustness. Leveraging TAMP's abstract states, we structure task decomposition and skill generation to reduce LLM hallucinations. We also define metric functions from TAMP predicates to enable LLMs to generate dense rewards for fast policy learning. To accelerate skill acquisition, trained policies are stored in a skill library, and relevant skills are retrieved via semantic similarity of LLM encoder features to warm-start training.

This creates a virtuous cycle between planning and skill learning, supporting efficient continual robot learning.

To summarize, our key contributions include: 1) using LLMs for continual skill learning via automatic task decomposition, skill creation, and dense reward generation; 2) leveraging structured state representations to regularize LLM content generation; 3) improving skill acquisition by maintaining and reusing a skill library; 4) demonstrating through extensive evaluation on six challenging simulated tasks from two benchmarks that our framework outperforms prior methods and supports real-world deployment, paving the way for lifelong robot learning in complex environments.

2 Related Work

2.1 LLMs for Reward Design

Designing effective reward functions has long been a challenge in reinforcement learning [39]. Recent studies have investigated using LLMs to generate reward functions [25, 26, 42–44]. While these methods have demonstrated potential in learning short-horizon skills, their scalability to complex, long-horizon tasks remains uncertain. Free-form LLM-based reward generation is prone to hallucinations, often producing syntax errors and irrelevant content. Moreover, approaches such as Eureka [25] are time-intensive, requiring multiple rounds of policy training and evaluation to refine the reward functions for each individual skill, making them impractical for long-horizon task learning. Zeng et al. [44] propose a method that iteratively refines reward functions based on feedback from the task learning process. However, this trial-and-error approach can result in delayed convergence and fluctuating performance, hindering consistent improvements. We address these limitations by parameterizing object relationships with metric functions, using LLMs to compose them for efficient, automated dense reward generation.

2.2 LLMs for Planning and Decision Making

In recent years, LLMs have been widely explored for robot planning [2, 13–15, 23, 33] and decision-making [20, 21, 38]. These studies demonstrate LLMs' ability to understand complex commands, create task plans, and translate natural language into executable actions. For instance, SayCan [2] uses LLMs to generate task plans, grounding each skill with learned control policies, while ProgPrompt [38] prompts LLMs to generate executable programs that invoke function calls for robot tasks. Researchers have also explored using LLMs to solve PDDL planning problems [37], showing that LLMs can act as generalized planners by generating efficient programs for tasks within a domain. This finding informs our approach. Unlike prior methods that generate skill sequences or programs, we use LLMs' semantic knowledge to construct planning domains and guide skill learning in TAMP, enabling more efficient and robust handling of long-horizon tasks.

2.3 TAMP and Learning for TAMP

Task and Motion Planning (TAMP) [9, 16] offers a robust framework for tackling long-horizon tasks in structured environment settings by decomposing complex planning problems into a sequence of symbolic-continuous subtasks, which simplifies the optimization for finding solutions. However, traditional TAMP systems require manually defined planning domains, which depend on extensive expert knowledge. To reduce this burden, recent works [18, 28, 35, 45] have integrated learning into TAMP, including efforts to learn low-level skill policies. Mandlekar et al. [27] proposed a TAMP-gated control mechanism that selectively transfers control between a human teleoperator and the robot, using the collected data to train skill policies and improve the TAMP system. LEAGUE [4] takes a different approach by leveraging the symbolic interface of task planners to guide RL-based skill learning, creating abstract state spaces that enable skill reuse and improve scalability for long-horizon tasks. Despite these advances, task decomposition still relies on handcrafted symbolic planners. Besides using symbolic planners for skill planning, researchers have also explored learned planners or heuristics to compose skills [29, 30, 46], which typically rely on task-specific training data, collected either via trial-and-error interaction or expert demonstrations. In this work, we utilize LLMs to integrate TAMP planning with skill learning, enabling robots to tackle long-horizon tasks more efficiently with minimal human input.

3 Background and Problem Setting

Planning domain and skill representations. We focus on deterministic, fully observed tasks that can be described in PDDL [8], with object-centric states, continuous actions, and a known transition function. Formally, an environment can be characterized by a tuple $\langle \mathcal{O}, \Lambda, \Psi, \Omega, \mathcal{G} \rangle$. Each object entity $o \in \mathcal{O}$ within the environment (e.g., `peg1`), possesses a specific type $\lambda \in \Lambda$ (such as `peg`) and a tuple of $\texttt{dim}(\lambda)$-dimensional features containing object state information such as pose and size. The environment state $x \in \mathcal{X}$ is a mapping from object entities to features: $x(o) \in \mathbb{R}^{\texttt{dim}(\texttt{type}(o))}$. Predicates Ψ describe the relationships among objects in the environment. Each predicate ψ (e.g., `Holding(?object:peg)`) is characterized by a tuple of object types $\langle \lambda_1, ..., \lambda_m \rangle$ and a binary classifier that determines whether the relationship holds: $c_\psi : \mathcal{X} \times \mathcal{O}^m \rightarrow \{True, False\}$, where each substitute entity $o_i \in \mathcal{O}$ is restricted to have type $\lambda_i \in \Lambda$. Each binary classifier is constructed through evaluating a set of metric functions $\mathcal{F}_\psi = \{f_i\}$ related to this predicate: $c_\psi \triangleq \bigwedge_{f_i \in \mathcal{F}_\psi} \mathbb{1}[f_i \leq \epsilon_i]$, where each metric function f_i (e.g., $\texttt{xy_dis}(\cdot,\cdot)$) outputs a real number to quantify the relationships among the query objects (e.g., the distance between two object centers projected onto the xy-plane). Evaluating a predicate on the state by substituting corresponding object entities will result in a ground atom (e.g., `Holding(peg1)`).

A task goal $g \in \mathcal{G}$ is represented as conjunction over a set of ground atoms (e.g., `In(peg1,hole1)` $\wedge$ `In(peg2,hole2)`), where a symbolic state x_Ψ can be obtained by evaluating a set of predicates Ψ and dropping all negative ground atoms.

Each lifted symbolic operator $\bar{\omega} \in \bar{\Omega}$ is defined by a tuple $\langle \texttt{Par}, \texttt{Pre}, \texttt{Eff}^+, \texttt{Eff}^- \rangle$, where `Pre` denotes the precondition of the operator, $\texttt{Eff}^+$ and $\texttt{Eff}^-$ are lifted atoms that describe the expected effects (changes in conditions) upon successful skill execution. `Par` is an ordered parameter list that defines all object types used in `Pre`, $\texttt{Eff}^+$, and $\texttt{Eff}^-$. An example of `Pick` operator can be defined as:

```
Pick(?object)
  PAR:  [?object: peg]
  PRE:  {HandEmpty(), OnTable(?object)}
  EFF-: {HandEmpty(), OnTable(?object)}
  EFF+: {Holding(?object)}
```

A ground skill operator $\underline{\omega}$ substitutes lifted atoms with object instances: $\underline{\omega} = \langle \bar{\omega}, \delta \rangle \triangleq \langle \underline{\texttt{Pre}}, \underline{\texttt{Eff}^-}, \underline{\texttt{Eff}^+} \rangle$, where $\delta : \Lambda \rightarrow \mathcal{O}$. A symbolic task plan is a sequence of ground operators that, when executed successfully, leads to an environment state that satisfies the task goal.

MDP. Learning the grounded low-level skills π for any symbolic operators ω can be formulated as a Markov Decision Process (MDP) denoted by $\langle \mathcal{X}, \mathcal{A}, \mathcal{R}(x,a), \mathcal{T}(x'|x,a), p(x^{(0)}), \gamma \rangle$, with continuous state space $\mathcal{X}$, continuous action space $\mathcal{A}$, reward function $\mathcal{R}$, environment transition model $\mathcal{T}$, distribution of initial states $p(x^{(0)})$, discount factor γ. The objective for RL training is to maximize the expected total reward J of the policy $\pi(a|x)$ that the agent employs to interact with the environment:

$$J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t \mathcal{R}(x^{(t)}) \right]. \tag{1}$$

Problem setting. Our setting is inspired by prior work on operator invention for bi-level planning [5, 36]. However, we assume only a small set of predefined predicates and metric functions. Given a task goal g, we automate operator discovery for decomposing long-horizon tasks and learn primitive skill policies to achieve the resulting subgoals. Each lifted operator $\bar{\omega}$ is associated with a single skill policy π, shared across all its groundings during execution.

4 Method

We seek to achieve automatic skill abstraction and continual learning in long-horizon tasks by leveraging the strong priors in LLMs. Our primary contribution is an integrated pipeline (Fig. 1) that uses LLMs for task decomposition and planning (Sec. 4.1), and constructs dense rewards for skill learning (Sec. 4.2). We describe how to accelerate learning of new skills in lifelong learning scenarios in Sec. 4.3.

4.1 Task Decomposition and Skill Creation

To reduce the domain knowledge needed for designing valid skills in TAMP, we propose leveraging the rich semantic knowledge of LLMs to decompose tasks into reusable, elementary skills. To address the hallucination issue—where generated task plans may overlook constraints between adjacent skills and "hallucinate" impossible effects—we incorporate structural information from TAMP system, including available predicates and the task goal description, as prompts to regularize the outputs. An example of operator generation is shown below:

```
Task: Generate preconditions and effects for the "
    Pick" action by selecting from the set of all
    available predicates. Ensure the output
    follows the format [Predicate1, Predicate2,
    ...].
Important: Each predicate is preceded by crucial
    comments in the code - these comments must be
    retained and considered carefully while
    generating the preconditions and effects.
- Action: "Pick"
- All predicates: {"Holding", "OnTable", "In",
    ...}
- The code of those predicates are shown below:
...
```

Plan verification. Although the availability of predicates simplifies the skill creation process, the generated operators may occasionally contain irrelevant preconditions and effects. To enhance the accuracy of task decomposition, we validate the generated symbolic operators using the A* algorithm to produce the symbolic task plan via deterministic search. To plan for a task goal g, we first evaluate the predicates on the current environment state x, yielding the corresponding symbolic state x_Ψ. We then ground each lifted operator $\bar{\omega} \in \bar{\Omega}$ by substituting object entities in the environment in preconditions and effects, leading to ground operators $\underline{\omega} \triangleq \langle \underline{\texttt{Pre}}, \underline{\texttt{Eff}^-}, \underline{\texttt{Eff}^+} \rangle$ that support operating with symbolic states. A ground operator is considered executable only when its preconditions are satisfied: $\underline{\texttt{Pre}} \subseteq x_\Psi$. The operators induce an abstract transition model $T(x_\Psi, \underline{\omega})$ that allows planning in symbolic space:

$$x'_\Psi = T(x_\Psi, \underline{\omega}) \triangleq (x_\Psi \setminus \underline{\texttt{Eff}^-}) \cup \underline{\texttt{Eff}^+}. \tag{2}$$

Compared to directly using LLMs for task plan generation, incorporating an A* planner provides feedback by evaluating the proposed operators and verifying whether they are sufficient to reach the task goal. If the search fails, unreachable operators are identified and used to guide LLM-based regeneration, enabling the LLM to produce correct operators with fewer attempts.

4.2 Reward Generation and Skill Learning

Once the planning domain is established, and symbolic skills can be planned for specific task goals, we can proceed to learn the corresponding low-level skill policies via deep reinforcement learning. Dense rewards are crucial for reinforcement learning, but human-crafted dense rewards require considerable effort. It presents challenges for long-horizon tasks with

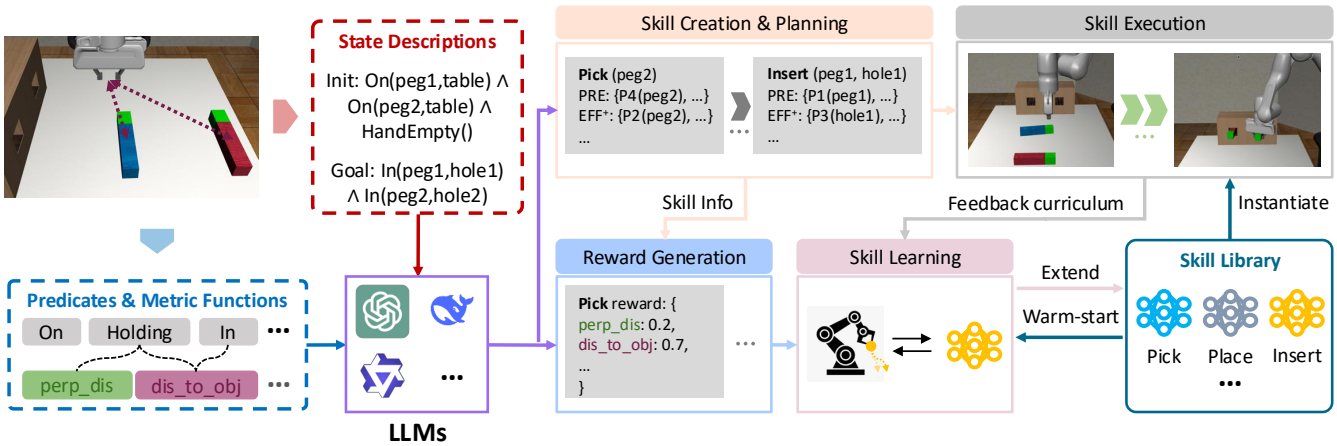


Figure 1. Framework Overview. Our framework automates task decomposition, skill creation, and dense reward generation by creating a virtuous cycle between planning and skill learning. Execution failures reveal skills needing improvement, while a skill library enables warm-starting new skills using semantic similarity.

various skills and lifelong learning scenarios where agents are exposed to diverse new tasks. We therefore consider using LLMs for automating reward generation.

Reward construction with metric functions. LLM-generated reward functions often contain imprecise code or syntax errors [25, 42, 43], requiring iterations or human feedback. Our key insight is that predicate-derived metric functions in TAMP encode object relationships, providing structural regularization on the generated content. Each metric function $f \in \mathcal{F}$ defines a quantitative relationship between objects: $f : \mathcal{X} \times \mathcal{O}^m \to \mathbb{R}$. For example, `dis_to_obj(robot, peg1)` defines the distance between the robot's gripper and the peg1 object. Instead of generating code from scratch, we simplify the problem by using LLMs with extracted skills definition from structural skill operators to select metric functions and their corresponding weights to construct dense rewards. This strategy improves the success rate of generating usable reward functions by constraining the LLMs' solution space, thereby reducing irrelevant or erroneous outputs. Additionally, the semantic information from metric functions aids the LLMs in composing reward functions that better align with task objectives.

To better harness the LLMs' proficiency in code interpretation and task comprehension, for each skill operator $\underline{\omega}$, we provide the LLM with the source code of metric functions along with generated skill operators. An examplar prompt for reward generation can be found below:

```
Task: Design a reward function for training "Pick"
     skill using RL based on the template.
- Skill: "Pick"
- Objects: ["peg", ...]
- Metric functions: ["dis_to_obj", ...]
- The template is defined as:
"""
Reward Template:
{'reward_template': [[reward score, '
    metric_function(inputs)'], ...]}
Guidelines to complete the reward template:
Select Metric Functions: Choose functions that
    help in learning the skill (code provided
    below).
Assign Reward Scores: Give each chosen function a
    score (0.0 to 1.0), with higher scores
    indicating greater importance.
"""
- The code of metric functions are shown below:
...
```

In this example, the metric function `dis_to_obj` with its corresponding code provides semantic information that helps the LLMs understand its purpose is to move the gripper closer to an object. Subsequently, the LLM generates dense rewards $\mathcal{R}^D_{\underline{\omega}}$ by selecting a subset of relevant metric functions $\mathcal{F}_{\underline{\omega}} = \{f_1, f_2, \ldots, f_n\} \subseteq \mathcal{F}$ and assigning weights $\mathcal{V}_{\underline{\omega}} = \{v_1, v_2, \ldots, v_n\}$ to their normalized forms f_i^*, reflecting their relative importance to the reward. The dense reward is thus defined as:

$$\mathcal{R}^D_{\underline{\omega}}(x^{(t)}) = \sum_{i=1}^{n} v_i \, f_i^*(x^{(t)}, O_{\underline{\omega}}), \qquad v_i,\ f_i^*(\cdot) \in [0,1] \ \ \forall i. \quad (3)$$

where $O_{\underline{\omega}}$ defines a group of object entities relevant to operator $\underline{\omega}$.

To ensure the learned policy achieves the desired outcomes specified by the skill operators, we incorporate sparse rewards derived from predicates. The agent receives a maximal reward when all predicates that define the desired effect of the skill operator $\underline{\omega}$ are met:

$$\mathcal{R}^S_{\underline{\omega}}(x^{(t)}) = \begin{cases} 1 & \text{if } (\wedge\, c_{\underline{\psi^+}}(x^{(t)})) \wedge (\wedge\, \neg c_{\underline{\psi^-}}(x^{(t)})) \\ & \text{for } \underline{\psi^+} \in \underline{\texttt{Eff}^+} \text{ for } \underline{\psi^-} \in \underline{\texttt{Eff}^-} \\ 0 & \text{otherwise} \end{cases} . \quad (4)$$

This complements the dense rewards constructed by the LLM, promoting the development of more robust skills for long-horizon tasks and ensuring the expected outcomes are met to seamlessly transition to the next skill. Finally, the skill reward is constructed as: $\mathcal{R}_{\underline{\omega}}(x^{(t)}) = \max(\mathcal{R}^D_{\underline{\omega}}(x^{(t)}), \mathcal{R}^S_{\underline{\omega}}(x^{(t)}))$.

Skill learning with state abstraction. With the LLM constructed dense rewards, we can train a policy for each symbolic skill using RL. Since the precondition and effect of a ground operator $\underline{\omega}$ induce an effective abstraction of the environment, we can define a skill-relevant state space to prevent the learned policy from being influenced by task-irrelevant objects, thus enhancing its learning efficiency and generalization [4]: $\hat{x} = \{x(o) : o \in \mathcal{O}_{\underline{\omega}}\}$. The training objective for RL is therefore formulated as:

$$J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t \mathcal{R}_{\underline{\omega}}(x^{(t)}) + \alpha \mathcal{H}(\pi(\cdot|\hat{x}^{(t)})) \right] . \quad (5)$$

We use Soft Actor-Critic (SAC) [12] to optimize the skill policy, where $\mathcal{H}$ is the entropy term. Despite the use of dense rewards, RL exploration can be inefficient for learning complex motions [4]. Conversely, while motion planning from TAMP systems struggles with contact-rich manipulation, it is well-suited for handling free-space motions [27]. Building on this, we propose augmenting our policy with motion planner-based primitives. The key idea is to first use an off-the-shelf motion planner to approach the skill-relevant object (as specified by the skill operator) before initiating RL-based skill learning. For the motion planning target, we set the goal position 0.04 m above the object or placement position identified by the task planner. This strategy facilitates more efficient exploration while preserving the ability to learn closed-loop, contact-rich manipulation skills.

4.3 Continual Skill Acquisition through Integrated Planning and Skill Learning

So far, we have described how to leverage the rich semantic knowledge from LLMs to guide task decomposition, skill operators creation, and reward generation for continual skill learning. In this section, we describe how to learn skills within the context of a task planning system. This integrated planning and learning approach ensures that the learned skills are compatible with the planner, while continuously expanding the system's capability to solve more tasks.

Task planning and skill execution. For any given task goal g, we use A* planner with the generated operators to search a task plan, as described in Sec. 4.1. With the generated task plan, we sequentially invoke the corresponding skill π^* to reach the subgoal that complies with the effects of each skill operator $\underline{\omega}$ in the plan. We roll out each skill until it fulfills the effects of the operator or a maximum skill horizon H is reached. To verify whether the l-th skill is executed successfully, we obtain the corresponding symbolic state x^l_Ψ by parsing the ending environment state x^*. Then, the execution is considered successful only when the environment state x^* conforms to the expected effects: $T(x^{l-1}_\Psi, \underline{\omega}_l) \subseteq x^l_\Psi$. We track failed skills along with the corresponding initial simulator state s^* to inform the learning curriculum, using this information to reset the environment for focused practice on those failed skills.

Automated Curriculum. To efficiently acquire the necessary skills for a multi-step task, we utilize the task planner as an automated curriculum, which enables progressive skill learning. The core idea is to use already proficient skills to achieve the preconditions of those skills which are still required further learning. At a high level, we iteratively alternate between task planning and skill learning until convergence. We track skill failures during executions and apply strict scheduling criteria: a skill is prioritized for further learning (Sec. 4.2) whenever it fails during rollouts. The algorithm is sketched in Alg. 1. Notably, we share the replay buffers across different skill instances (e.g., `Pick(peg1)` and `Pick(peg2)`) that correspond to the same lifted operator. This allows relevant trajectories to be reused, which enhances both learning efficiency and generalization.

Accelerate learning with skill library. Efficient learning of complex tasks requires the continual adaptation of acquired skills to new tasks, even across different domains. Our insight is that semantically similar skill operators often share underlying low-level behaviors. For instance, opening a refrigerator and opening a door may involve similar actions and interactions. As a result, policy models from previously learned skills can serve as effective initializations, providing a warm start for learning new skills in novel situations.

Based on this, we propose to maintain a semantic skill library, where each element is a key-value pair. The key represents the semantic embedding of the skill's symbolic definition, which is extracted by a pre-trained LLM encoder (specifically, OpenAI's text-embedding-3-large): $z = \Phi(\omega) \triangleq \Phi(\langle \texttt{Pre}, \texttt{Eff}^-, \texttt{Eff}^+ \rangle)$. The value corresponds to the neural network weights of the associated policy.

To leverage the models stored in this library, for any new skill that needs to be acquired, we first extract its feature embedding z'. We then identify the most similar skill in the library using cosine similarity and use the corresponding weights to initialize the new skill, which provides a warm start for continual training. Each newly learned skill is subsequently added to the library, expanding it to speed up the learning process.

Algorithm 1 SKILLACQUISITION

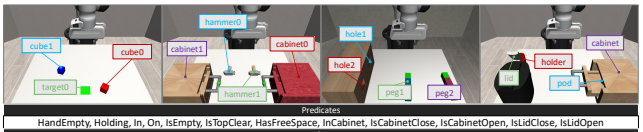
input: env (task environment), g (symbolic goal), $\bar{\Psi}$ (state predicates), $\bar{\Omega}$ (operators by LLM), $\mathcal{R}$ (reward functions by LLM), Π (skill library), Φ (LLM encoder)
start:
$O, x^{(0)}, s^{(0)} \leftarrow$ `env.get_state()`
$x_{\Psi}^{(0)} \leftarrow \text{PARSE}(x^{(0)}, O, \bar{\Psi})$ ▹ continuous state to symbolic state
$\underline{\Omega} \leftarrow \text{GROUND}(O, \bar{\Omega})$ ▹ get grounded operators
$[\underline{\omega}_1, ..., \underline{\omega}_L] \leftarrow \text{SEARCH}(x_{\Psi}^{(0)}, g, \underline{\Omega})$ ▹ found plan with length L
$t \leftarrow 0$
while *Not Converged* **do**
  `env.set_state(`$s^{(0)}$`)`
  $[\pi_1^{(t)}, ..., \pi_L^{(t)}] \leftarrow \text{RETRIEVE}([\underline{\omega}_1, ..., \underline{\omega}_L], \Pi)$ ▹ warm-start initialization
  $\mathcal{D} \leftarrow \text{ROLLOUT}(\text{env}, \underline{\Omega}, [\pi_1^{(t)}, ..., \pi_L^{(t)}])$ ▹ failed skills and simulator states
  if $\mathcal{D} = \emptyset$ **then**
    break
  end if
  for $s, \underline{\omega} \leftarrow \mathcal{D}$ **do**
    `env.set_state(`s`)`
    $\pi_i^{(t)} \leftarrow \Pi[\Phi(\underline{\omega})]$
    $\pi_i^{(t+1)} \leftarrow \text{SAC}(\text{env}, \pi_i^{(t)}, \underline{\omega}, \mathcal{R}_{\underline{\omega}})$ ▹ RL training
    $\Pi \leftarrow \Pi \cup \{\Phi(\underline{\omega}) : \pi_i^{(t+1)}\}$
  end for
  $t \leftarrow t + 1$
end while
return Π

5 Experiment

In this section, we show that LG-SAIL autonomously learns long-horizon tasks and outperforms prior state-of-the-art methods. We demonstrate its ability to adapt skills to novel tasks and domains—crucial for continual learning—and validate our planning and reward designs via ablations. Finally, we deploy the system on a physical robot platform and present both quantitative and qualitative results in the real world.

5.1 Experimental Setup

We conduct experiments in six simulated domains. The domain STACKATTARGET, STOWHAMMER, PEGINHOLE, and MAKECOFFEE are from LEAGUE [4], which involve long-horizon reasoning and contact-rich manipulation. The task setups, predicates, and our generated operators and task plan are shown in Fig. 2. To further evaluate the framework's ability to efficiently adapt learned skills to novel tasks, we include the LIBERO benchmark [24], a benchmark in lifelong robot learning that features diverse objects and long-horizon tasks. We test our framework on both LIBERO-SPATIAL and LIBERO-OBJECT. The LIBERO-SPATIAL set comprises 10 tasks with varying scene configurations, where the robot must pick up a target bowl and place it on the goal plate. The LIBERO-OBJECT set includes 10 tasks involving different objects with diverse shapes, which requires the robot to grasp the target object and place it in a basket. The setups and our generated operators are shown in Fig. 3.



Predicates			
HandEmpty, Holding, In, On, IsEmpty, IsTopClear, HasFreeSpace, InCabinet, IsCabinetClose, IsCabinetOpen, IsLidClose, IsLidOpen			
Metric functions			
is_grasped, dis_to_obj, xy_dis, z_dis, parallel_dis, perpendicular_dis, norm_angle			
Training goal			
On(cube1,cube0) ∧ On(cube0,target0)	In(hammer0,cabinet0) ∧ In(hammer1,cabinet1) ∧ IsCabinetClose(cabinet0) ∧ IsCabinetClose(cabinet1)	In(peg1,hole1) ∧ In(peg2,hole2)	In(pod,holder) ∧ IsLidClose(lid) ∧ IsCabinetClose(cabinet)
Our generated operators and task plan			
Pick(cube0), Place(cube0,target0), Pick(cube1), Place(cube1,cube0)	Open(cabinet0), Pick(hammer0), Place(hammer0,cabinet0), Close(cabinet0), Open(cabinet1), Pick(hammer1), Place(hammer1,cabinet1), Close(cabinet1)	Pick(peg2), Insert(peg2,hole2), Pick(peg1), Insert(peg1,hole1)	Open(cabinet), Pick(pod), Insert(pod,holder), CloseLid(lid), Close(cabinet)

Figure 2. LEAGUE Tasks [4]. Illustration of task setups for STACKATTARGET, STOWHAMMER, PEGINHOLE, and MAKECOFFEE.

Predicates	
HandEmpty, Holding, In, On	
Metric functions	
is_grasped, dis_to_obj, xy_dis, z_dis	
Training goal	
On(bowl0,plate0)	In(object0,basket0)
Our generated operators and task plan	
Pick(bowl0), Place(bowl0,plate0)	Pick(object0), Place(object0,basket0)

Figure 3. LIBERO [24]. Tasks setup for LIBERO-OBJECT and LIBERO-SPATIAL.

All environments are built on the MuJoCo engine [40]. We use a Franka robotic arm, controlled at 20Hz with an operational space controller (OSC), providing 5 degrees of freedom: end-effector position, yaw angle, and gripper position.

5.2 Quantitative Evaluation

To assess performance, we compare LG-SAIL against strong baselines for learning long-horizon tasks in STACKATTARGET, STOWHAMMER, and PEGINHOLE:

- **RL (SAC)** - We utilize Soft Actor-Critic (SAC) [12] for vanilla RL baseline;
- **Curriculum RL (CRL)** - This baseline follows the same curriculum strategy as prior works [34, 41], starting training from near-success states and gradually shifting the reset states back to the initial task states. We use the same model as **RL (SAC)** for fair comparison;
- **Hierarchical RL (HRL)** - This baseline employs HRL designs [6, 31], where a high-level controller composes parameterized skill primitives. Our implementation builds on MAPLE [30] and uses the oracle task

plan to identify target objects for affordance definition.

- **LEAGUE** [4] - This baseline reflects the recent trend of integrating TAMP with skill learning. We utilize it for this experiment due to its superior performance on long-horizon tasks.

To ensure fair comparison, we follow experiment settings in LEAGUE [4], and we adopt task progress as evaluation metric, which is defined as the summed reward of all task stages and normalized to [0, 1]. The results are presented in Fig. 4.

Our framework efficiently and autonomously learns long-horizon manipulation tasks. By just providing predicates, metric functions, and task goals of different domains (shown in Fig. 2), our framework autonomously achieves task decomposition, skill creation, reward generation, and integrates TAMP with skill learning. Additionally, we found that methods incorporating skill abstraction and planning (i.e., LG-SAIL and LEAGUE [4]) significantly outperform other baselines, underscoring the importance of explicit skill reuse in multi-step tasks with repeating structures. Notably, LG-SAIL demonstrates slightly higher learning efficiency compared to LEAGUE [4], potentially reflecting the advantages of LLM-generated rewards over handcrafted ones, as supported by recent literature [25, 26].

5.3 Validating Skill Generalization and Adaptation

We design experiments to evaluate the system's continual learning capability by constructing tasks that require varying levels of adaptation.

Generalizing to new goals. In addition to training goals for PegInHole and StowHammer, we directly evaluate our framework on new goals without further learning. For StowHammer, the first test goal is to swap the hammer-cabinet mapping, while the second goal is to place `hammer1` into `cabinet0` while keeping `cabinet1` open. For PegInHole, the first test goal is to swap the peg-hole mapping, and the second goal is to insert `peg1` into `hole2`. The results are presented in Tab. 1.

Table 1. Task goal generalization. We evaluate RL (SAC) and LG-SAIL on novel task goals in the StowHammer and PegInHole domains under zero-shot settings.

Domain	Method	Training Goal	Test Goal 1	Test Goal 2
StowHammer	RL (SAC)	0.15 ± 0.02	0.07 ± 0.02	0.09 ± 0.01
	LG-SAIL	**0.96 ± 0.15**	**0.91 ± 0.09**	**0.71 ± 0.28**
PegInHole	RL (SAC)	0.19 ± 0.03	0.09 ± 0.01	0.10 ± 0.01
	LG-SAIL	**0.92 ± 0.11**	**0.59 ± 0.14**	**0.93 ± 0.06**

We observe that LG-SAIL exhibits minimal performance drop when generalizing to new goals within the domain without additional training, demonstrating strong compositional generalization capabilities.

Adapting to new objects and scenes. We aim to evaluate LG-SAIL's ability to adapt learned skills to novel situations using the LIBERO [24] benchmark. The LIBERO-Object set requires the robot to pick up various target objects and place them into a basket, testing its ability to adapt to new object shapes and poses. The LIBERO-Spatial set features diverse scene layouts where the robot must place a target bowl on a plate, emphasizing generalization to different object poses and scene configurations. We sequentially test LG-SAIL on tasks from both LIBERO-Object and LIBERO-Spatial, with results shown in Fig. 5. Interestingly, while LG-SAIL takes some time to learn the initial task, as more skills are accumulated, the learning of subsequent tasks accelerates significantly, demonstrating its strong potential for continual and lifelong learning.

Learning in new domains. Another advantage of LG-SAIL is its ability to reuse learned models from other domains to warm-start the learning of similar skills in new domains.

To validate this, we evaluate LG-SAIL on MakeCoffee. LG-SAIL retrieves skills `Open`, `Close`, `Pick`, and `Place` from StowHammer in the skill library using semantic embeddings and uses these models to initialize skill policies, leading to improved learning efficiency compared to training from scratch (Fig. 6). Compared to `League`, LG-SAIL can retrieve skills that are similar but not identical by measuring semantic distances over symbolic operators—such as warm-starting the novel skill `Insert` with `Place` from StowHammer—a transfer that is difficult to specify manually. In addition, this task again demonstrates that LLM-generated rewards provide more informative supervision than human-designed rewards.

5.4 Ablation Study

We evaluate various design choices for task planning and reward generation in LG-SAIL.

Task planning. We compare three design choices of using LLMs for skill generation and planning, we use GPT-4 [1] in our experiments:

- **LLM+A***: This variant is adopted in LG-SAIL, where LLMs are used for skill generation, and A* search is utilized to find task plan based on the generated operators. If no valid task plan is found, the skill generation process is re-invoked to refine the operators.
- **LLM**: Similar to prior works on LLM-based planning [7, 17], this variant uses LLMs for both skill operator generation and task planning. If the generated plan is not executable, the process is re-invoked to produce a new plan.
- LLM Planner w/o Replanning (**LLM 1-shot**): This variant skips task plan validation and invokes skill generation and task planning only once.

We evaluate StowHammer across four aspects: (1) success rate of generating skill operators with correct preconditions and effects for connecting other skills, (2) success rate of

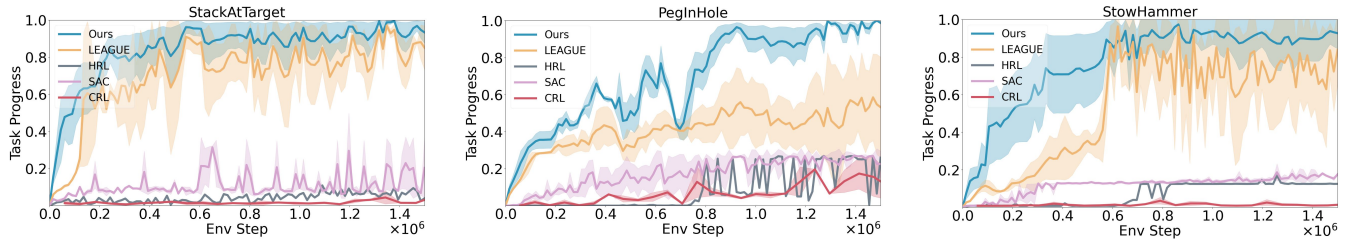


Figure 4. Baseline comparison. We compare our framework with other baselines across three task domains. The plot illustrates the average task progress during evaluation over the training phase, measured as the sum of rewards for each successfully execution in the task plan, normalized to 1. The shaded area represents the standard deviation for 5 random seeds.

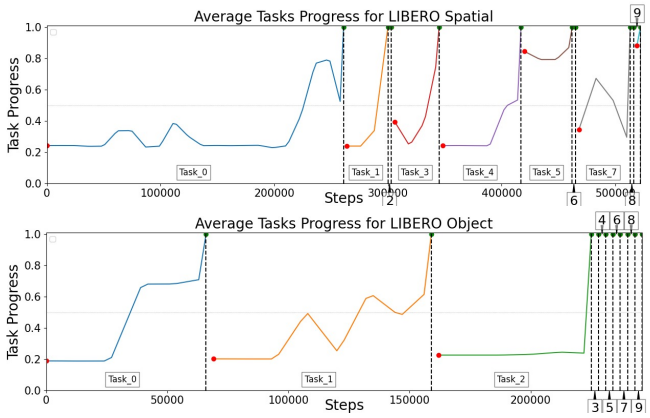


Figure 5. Continuous skill adaptation and learning. The learning process is shown for both LIBERO-Object and LIBERO-Spatial tasks.

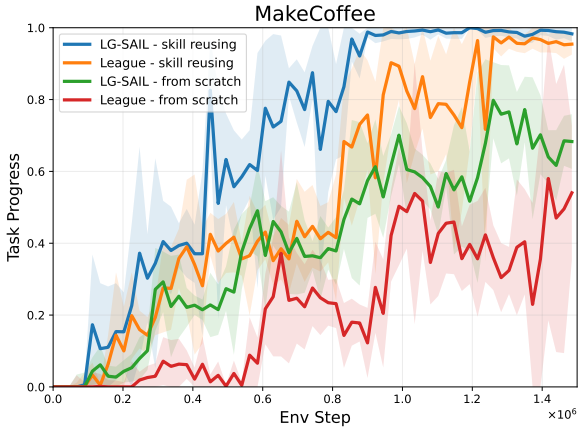


Figure 6. Skill adaptation in new domains. For MakeCoffee, we compare learning from scratch and adapting skills learned in the StowHammer domain with both LG-SAIL and League.

generating valid plans conditioned on correctly generated skills, (3) overall task success rate per method (for **LLM+A*** and **LLM**, skill generation is retried until a valid plan is found), and (4) average number of attempts. Each variant runs until 25 correct plans are generated.

Results are in Tab. 2. We found that both design choices with a replanning strategy achieved 100% success rate, while the **LLM 1-shot** variant only reached a 36% success rate. Additionally, the **LLM + A*** approach guarantees a 100% success rate if correct skills are generated, effectively reducing LLM-based planning hallucinations and minimizing the number of replanning attempts.

Reward generation. We validate the impact of full metric functions on reward generation by comparing our approach (**Full MF**) to an ablated variant (**W/O MF**) that provides only metric function's header to the LLM, omitting both the code implementation and descriptive comments. We use GPT-4 [1] in our experiments. Both are tested on StowHammer over 25 trials, evaluating reward validity for each skill. Following Xie et al. [42], we classify errors into four types: (1) *Class attribute misuse* - incorrect object selection for metric functions, (2) *Attribute hallucination* - referencing non-existent entities, (3) *Syntax/format errors* - structural mistakes, and (4) *Metric selection errors* - inappropriate function choices for rewards. Success is based on error-free reward generation, not its impact on policy optimization.

Tab. 3 presents the reward generation success rates for StowHammer. Our method achieves 92% overall success, with 96% for `Pick` and `Place`, and 100% for `Open` and `Close`. The low error rate underscores its reliability for long-horizon tasks and potential for lifelong learning.

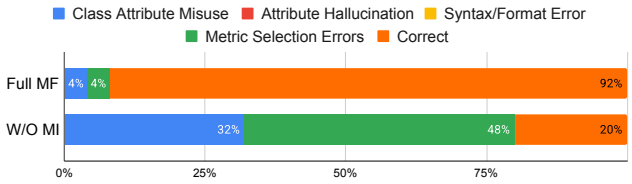


Figure 7. Error breakdown. We show the error distribution of reward generation for StowHammer.

Fig. 7 illustrates the error distribution across reward generation strategies. In LG-SAIL, using improved prompts and detailed metric functions reduces attribute/object selection errors from 80% to 8%. This emphasizes that structured information, like quantified object relationships, enhances LLMs' scene interpretation and reward accuracy.

General Compatibility with Alternative LLMs.

Table 2. Task Planning Comparisons on StowHammer. We compare LLM-native baselines (**left**) and alternative LLMs within our framework (**right**). The task planning success rate is calculated based on trials where the skills are correctly generated. All models in our framework achieve a 100% end-to-end success rate via replanning, as A* guarantees plan possibility under the generated operators. Stronger base models require fewer attempts on average, indicating favorable scaling behavior.

	LLM-native Baselines		Our Method w/ Alternative LLMs (LLM+A*)				
Metric	**GPT-4**	**GPT-4 1-shot**	**GPT-4**	**GPT-5.2**	**Llama-3.3-70B**	**Qwen3-VL-235B**	**DeepSeek-V3**
Skill Generation Success Rate (↑)	59.3%	52.0%	**61.0%**	79.37%	90.09%	92.59%	98.04%
Task Planning Success Rate (↑)	78.1%	69.2%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Full Design Success Rate (↑)	**100.0%**	36.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Average Attempts (↓)	2.16	N/A	1.64	1.26	1.11	1.08	**1.02**

Table 3. Reward Generation Comparisons on StowHammer. We compare different reward construction methods and the compatibility of our **Full MF** approach across alternative foundation models. Results represent the success rate over 25 trials.

Setting	LLM	Overall	Pick	Place	Open	Close
W/O MF	GPT-4	20.0%	40.0%	36.0%	44.0%	48.0%
Full MF	GPT-4	92.0%	96.0%	96.0%	**100.0%**	**100.0%**
	Llama-3.3	97.0%	96.0%	92.0%	**100.0%**	**100.0%**
	DeepSeek-V3	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
	Qwen-3	99.0%	**100.0%**	96.0%	**100.0%**	**100.0%**
	GPT-5.2	98.0%	**100.0%**	**100.0%**	92.0%	**100.0%**

To assess compatibility with open-source LLMs, we evaluate four state-of-the-art foundation models—GPT-5.2, DeepSeek-V3 [22], Qwen3-VL-235B [3], and Llama-3.3-70B [10]—on both task planning and reward generation.

For task planning and reward generation, we evaluate: (1) skill generation success rate, (2) *task planning success rate*, i.e., whether A* can find a plan whenever one exists under the generated operators, (3) full design task success rate, (4) average attempts, and (5) reward generation success using **Full MF**. For (1)–(4), each variant runs until 100 correct plans are generated; for (5), each skill is evaluated over 25 trials.

Results in Tab. 2 and Tab. 3 show broad compatibility across all models. Initial operator generation success rates range from 79.37% to 98.04%, with only 1.02–1.26 attempts on average. With replanning, A* guarantees *plan possibility*, yielding 100% task success across all models. Reward generation is similarly robust, achieving over 97% zero-shot success on average without model-specific tuning.

These results also suggest a scaling trend: stronger base models improve initial skill generation and reduce replanning, making the framework efficient as LLMs advance.

5.5 Real World Results

We demonstrate the transfer of our simulation-trained LG-SAIL system to two real-world tasks: StackAtTarget, where the robot stacks two cubes in a target region in a specified order, and SortBowl, which is the task in LIBERO Benchmark that involves placing a bowl on a target plate.

Our system employs a Franka Emika robotic arm and a Microsoft Azure Kinect camera for RGBD image capture. We use AprilTag [32] to detect the 6D poses of objects and perform state estimation to synchronize with the simulated environment. Skills generated by LG-SAIL in simulation are executed in the real world through open-loop control.

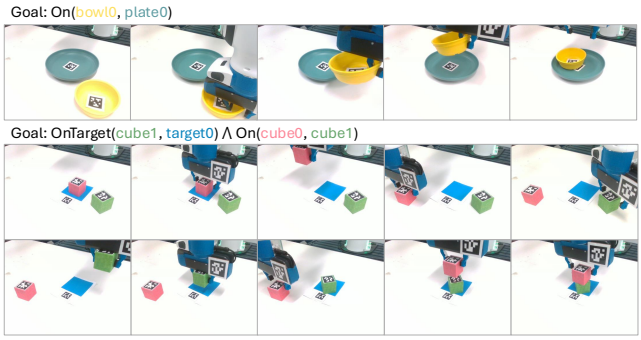


Figure 8. Real-world results. We demonstrate the deployment of LG-SAIL for real-world tasks.

We conduct 10 trials per task, with key execution frames in Fig. 8. While occasional failures occur due to occlusion-induced pose estimation errors and bowl slippage from insecure grasps, we achieve overall success rates of 0.8 and 1.0, highlighting the robustness of our system.

6 Conclusions and Limitations

We present LG-SAIL, a framework that integrates LLM-guided skill generation and learning with TAMP, enhancing automatic and continuous robot learning across various tasks. Our results show that LG-SAIL outperforms previous approaches by reducing human intervention and improving learning efficiency.

Despite strong performance on challenging long-horizon tasks, we observe common failure modes, including object slippage from unsecured grasps and unexpected contact behaviors. These issues could be mitigated by incorporating failure context into reward generation and by improving robustness during execution.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. In *arXiv*.

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do As I Can and Not As I Say: Grounding Language in Robotic Affordances. In *arXiv*.

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631* (2025).

[4] Shuo Cheng and Danfei Xu. 2023. LEAGUE: Guided Skill Learning and Abstraction for Long-Horizon Manipulation. *RA-L* (2023).

[5] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. 2022. Learning neuro-symbolic relational transition models for bilevel planning. In *IROS*.

[6] Murtaza Dalal, Deepak Pathak, and Ruslan Salakhutdinov. 2021. Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives. In *NeurIPS*.

[7] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement. In *IROS*.

[8] Maria Fox and Derek Long. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *JAIR* (2003).

[9] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. Integrated task and motion planning. *Annu. Rev. Control Robot. Auton. Syst.* (2021).

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[11] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*.

[13] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*.

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv* (2022).

[15] Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. 2024. CoPAL: corrective planning of robot actions with large language models. In *ICRA*.

[16] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2013. Integrated task and motion planning in belief space. *IJRR* (2013).

[17] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. 2024. Smart-llm: Smart multi-agent robot task planning using large language models. In *IROS*.

[18] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *JAIR* (2018).

[19] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *JMLR* (2016).

[20] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. 2022. Pre-trained language models for interactive decision-making. *NeurIPS* (2022).

[21] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2022. Code as Policies: Language Model Programs for Embodied Control. In *arXiv*.

[22] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[23] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. In *arXiv*.

[24] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. 2024. Libero: Benchmarking knowledge transfer for lifelong robot learning. *NeurIPS* (2024).

[25] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv* (2023).

[26] Yecheng Jason Ma, William Liang, Hungju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. 2024. DrEureka: Language Model Guided Sim-To-Real Transfer. In *RSS*.

[27] Ajay Mandlekar, Caelan Reed Garrett, Danfei Xu, and Dieter Fox. 2023. Human-in-the-loop task and motion planning for imitation learning. In *CoRL*.

[28] Michael James McDonald and Dylan Hadfield-Menell. 2022. Guided imitation of task and motion planning. In *CoRL*.

[29] Utkarsh Aashu Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. 2023. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning*. PMLR, 2905–2925.

[30] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. 2022. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *ICRA*.

[31] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. 2022. Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks. In *ICRA*.

[32] Edwin Olson. 2011. AprilTag: A robust and flexible visual fiducial system. In *ICRA*.

[33] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. 2023. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. In *CoRL*.

[34] Archit Sharma, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2021. Autonomous Reinforcement Learning via Subgoal Curricula. In *NeurIPS*.

[35] Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Learning Neuro-Symbolic Skills for Bilevel Planning. In *CoRL*.

[36] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. 2023. Predicate invention for bilevel planning. In *AAAI*.

[37] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *AAAI*.

[38] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using

large language models. In *ICRA*.

[39] Richard S Sutton. 2018. Reinforcement learning: An introduction. *A Bradford Book* (2018).

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *IROS*.

[41] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. 2023. Jump-start reinforcement learning. In *ICML*.

[42] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. 2023. Text2reward: Automated dense reward function generation for reinforcement learning. *arXiv* (2023).

[43] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. 2023. Language to rewards for robotic skill synthesis. *arXiv* (2023).

[44] Yuwei Zeng, Yao Mu, and Lin Shao. 2024. Learning Reward for Robot Skills Using Large Language Models via Self-Alignment. *arXiv* (2024).

[45] Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, and Ye Zhao. 2024. A Survey of Optimization-based Task and Motion Planning: From Classical To Learning Approaches. *arXiv* (2024).

[46] Zexin Zheng, Jia-Feng Cai, Xiao-Ming Wu, Yi-Lin Wei, Yu-Ming Tang, Ancong Wu, and Wei-Shi Zheng. 2025. imanip: Skill-incremental learning for robotic manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 13890–13900.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009