Continual Robot Learning via Language-Guided Skill Acquisition

Shuo Cheng*[1], Zhaoyi Li*[1], Kelin Yu*[2], Danfei Xu[1]

***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 the potential to learn fine-grained skills, but is based heavily on human-defined rewards and faces challenges with long-term goals. Task and Motion Planning (TAMP) is adept at handling long-horizon tasks but often needs tailored domain-specific skills, resulting in practical limitations and inefficiencies. To address these challenges, 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, skill creation, and dense reward generation to efficiently acquire 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.**

I. 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 [1, 2], they require carefully designed reward functions and often fail to achieve long-term goals in complex environments. Conversely, Task and Motion Planning (TAMP) [3] 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 [4–6]. 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 (shown in Fig. 1), a continuous learning paradigm for robot manipulation that leverages large language models (LLMs) for integrating TAMP and DRL. Our key insight is that LLMs, trained on web-scale data, provide common sense knowledge for task decomposition and skill creation, while TAMP's structured state representations (e.g., predicates) serve as strong priors to guide LLM reasoning, structuring task decomposition and mitigating hallucinations.

To summarize, our key contributions include: 1) Employing LLMs to facilitate continual skill learning through automatic task decomposition, skills creation, and dense reward generation; 2) Instead of free form code generation, we leverage structural state representation to ease and regularize LLMs content generation; 3) Improving the acquisition of new skills by maintaining a skill library to reuse learned skills during training; 4) By conducting comprehensive evaluation on six challenging simulated task domains from two benchmarks, we analyze the efficacy of our design and demonstrate the framework's superior performance, which opens the potential for lifelong robot learning in complex environments. We also show that the learned skills can be readily deployed on real world robot.

II. 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 a integrated pipeline (Fig. 1) that uses LLMs for task decomposition and planning (Sec. II-A), and constructs dense rewards for skill learning (Sec. II-B). We describe how to accelerate learning of new skills in lifelong learning scenario in Sec. II-C. Background and problem formulation are shown in Appendix. IV-I

A. 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 in Appendix IV-B.

Plan verification and error feedback. 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 A* algorithm to produce the symbolic task plan via

*Equally Contributed
[1]Georgia Institute of Technology
[2]University of Maryland, College Park

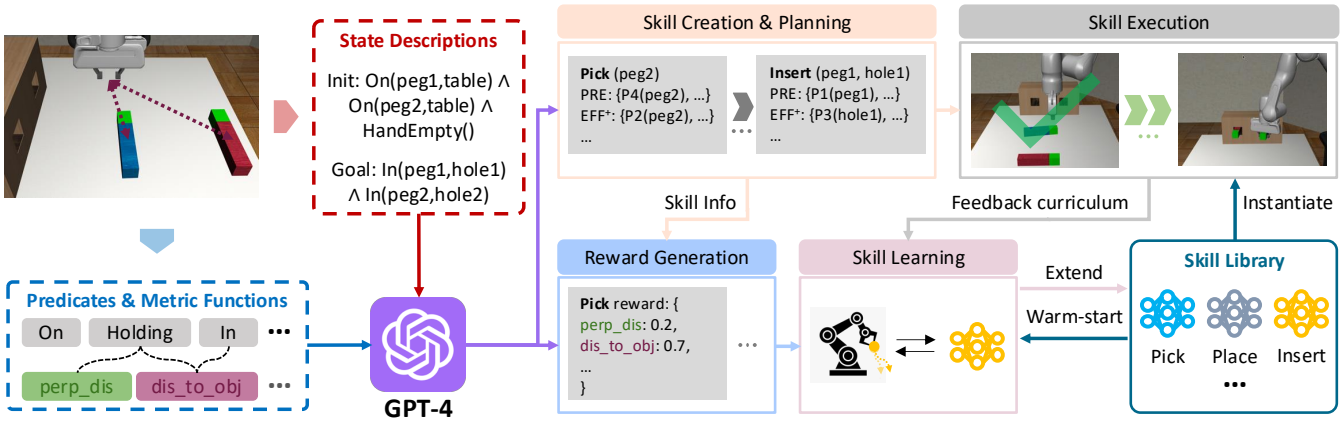

Fig. 1: **Framework Overview.** Our framework automates task decomposition, skill creation, and dense reward generation by creating a virtuous cycle between planning and skill learning. Execution failures reveal skills needing improvement, while a symbolic skill library enables warm-starting new skills using semantically related models.

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

Compared to directly using LLMs for task plan generation, an A$*$ planner can provide feedback by evaluating generated plan and generate usable operators with fewer attempts. Additionally, this approach verifies whether the current set of symbolic skills is sufficient to achieve the task goal. If verification fails, error information from the planner will be incorporated into the LLM prompt to regenerate the necessary skill operators.

B. 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 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 [7–9], which LLMs or human feedback for tuning. 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 spatial relationship between objects with normalization: $f : \mathcal{X} \times \mathcal{O}^m \to [0, 1]$. For example, `dis_to_obj(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 LLM solution space, thus reducing irrelevant or erroneous outputs. Additionally, the semantic information embedded in the 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 definitions of metric functions along with generated skill operators in the form of source code. In this work we use GPT-4 [10]. An examplar prompt for reward generation can be found in Appendix IV-C.

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: $\mathcal{R}^S_{\underline{\omega}}(x^{(t)}) = \begin{cases} 1 & \text{if } \bigwedge c_{\underline{\psi}}(x^{(t)}) \text{ for } \underline{\psi} \in \underline{\texttt{Eff}}^+ \\ 0 & \text{otherwise} \end{cases}$. 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 then train the policy corresponding to each symbolic skill with 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: $J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t \mathcal{R}_{\underline{\omega}}(x^{(t)}) + \alpha \mathcal{H}(\pi(\cdot | \hat{x}^{(t)})) \right]$. We adopt Soft Actor-Critic (SAC) [11] 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 [6]. Building on this, we propose augmenting our policy with motion planner-based transition 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 $0.04m$ above the object or placement position identified by the task planner. This approach accelerates exploration while maintaining the ability to learn closed-loop, contact-rich manipulation skills.

C. 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 to ensure 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. II-A. 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 keep track of the failed skills and the starting simulator info s^* to inform the learning curriculum.

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. II-B) 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 are 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's policy, 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 of future tasks.

III. Experiment

In this section, we demonstrate that LG-SAIL can autonomously learn long-horizon tasks and outperforms prior state-of-the-art methods. We showcase LG-SAIL's ability to adapt skills to novel tasks and domains, a critical aspect for continual and lifelong robot learning. Additionally, we validate our task planning and reward generation designs through ablation studies. Finally, we present quantitative and qualitative results for real-world tasks.

Details on evaluating the system's continual skill learning capability are provided in Appendix IV-E, along with ablation studies on our hybrid LLM-TAMP planning system and reward generation in Appendix IV-F.

A. 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 and predicates are shown in Fig. 3. To further evaluate the framework's ability to efficiently adapt learned skills to novel tasks, we include the LIBERO benchmark [12], 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 are shown in Fig. 4.

All environments are built on MuJoCo engine [13]. We use a Franka robotic arm, controlled at 20Hz with an operational

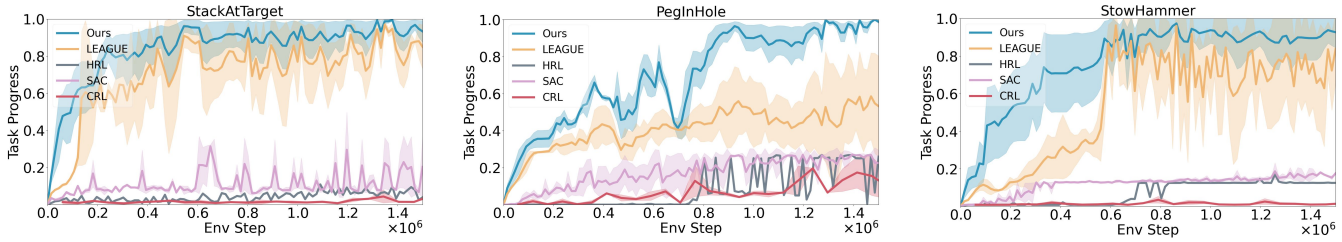

Fig. 2: **Baseline comparison.** We compare our framework with other baselines across three task domains. The plot illustrates the average task progress during evaluation over the training phase, measured as the sum of rewards for each successfully executed skill in the task plan, normalized to 1. The shaded area represents the standard deviation for 5 random seeds.

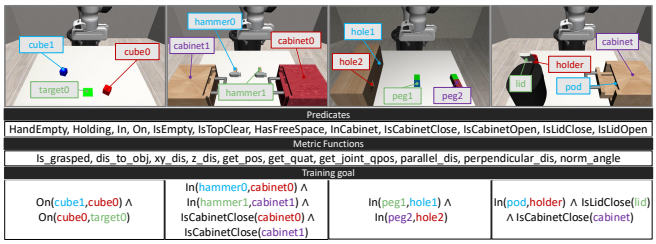

Fig. 3: **LEAGUE [4] Tasks.** Illustration of task setups for "`StackAtTarget`", "`StowHammer`", "`PegInHole`", and "`MakeCoffee`".

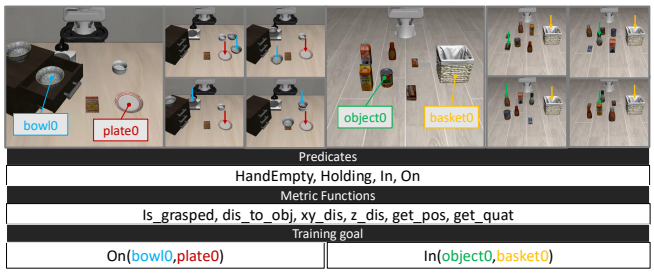

Fig. 4: **LIBERO [12] Tasks.** Illustration of task setups for "`LIBERO-Object`" and "`LIBERO-Spatial`".

space controller (OSC), providing 5 degrees of freedom: end-effector position, yaw angle, and gripper position.

B. Quantitative Evaluation

To assess performance, we compare LG-SAIL against strong baselines for learning long-horizon tasks in "`StackAtTarget`", "`StowHammer`", and "`PegInHole`", which are **RL (SAC)**, **Curriculum RL (CRL)**, **Hierarchical RL (HRL)**, and **LEAGUE**. Details about them are shown in Appendix IV-D.

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 in Fig. 2.

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. 3), 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 [8, 14].

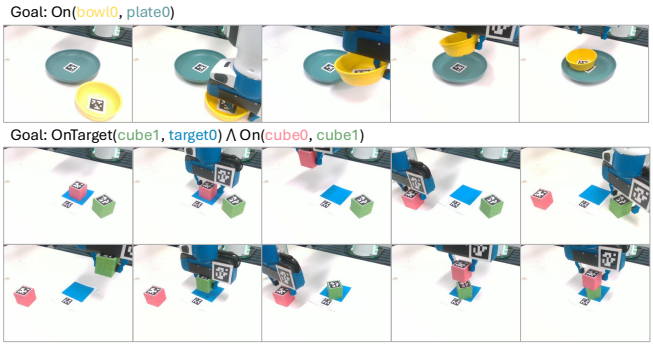

Fig. 5: **Real-world results.** We demonstrate the deployment of LG-SAIL for real-world tasks.

C. Real World Results

We demonstrate the transfer of the simulation-trained policy with our 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 involves placing a bowl on a target plate.

Our system uses a Franka Emika robotic arm and a Microsoft Azure Kinect camera for RGBD image capture. We use AprilTag [15] to detect the 6D poses and perform state estimation to match relevant objects states in simulated environment. Skills generated by LG-SAIL in simulation are executed in the real world via open-loop control.

We conduct 10 trials per task, with key execution frames in Fig. 5. While occasional failures occur due to occlusion-induced pose estimation errors and bowl slippage from insecure grasps, we achieve overall success rates of 80% and 100%, highlighting the robustness of our system.

IV. Conclusions

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.

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2016.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *ICRA*, 2017.

[3] C. R. Garrett *et al.*, "Integrated task and motion planning," *Annu. Rev. Control Robot. Auton. Syst.*, 2021.

[4] S. Cheng and D. Xu, "League: Guided skill learning and abstraction for long-horizon manipulation," *RA-L*, 2023.

[5] M. J. McDonald and D. Hadfield-Menell, "Guided imitation of task and motion planning," in *CoRL*, 2022.

[6] A. Mandlekar, C. R. Garrett, D. Xu, and D. Fox, "Human-in-the-loop task and motion planning for imitation learning," in *CoRL*, 2023.

[7] T. Xie *et al.*, "Text2reward: Automated dense reward function generation for reinforcement learning," *arXiv*, 2023.

[8] Y. J. Ma *et al.*, "Eureka: Human-level reward design via coding large language models," *arXiv*, 2023.

[9] W. Yu *et al.*, "Language to rewards for robotic skill synthesis," *arXiv*, 2023.

[10] J. Achiam *et al.*, "Gpt-4 technical report," in *arXiv*, 2023.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[12] B. Liu *et al.*, "Libero: Benchmarking knowledge transfer for lifelong robot learning," *NeurIPS*, 2024.

[13] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[14] Y. J. Ma *et al.*, "Dreureka: Language model guided sim-to-real transfer," in *RSS*, 2024.

[15] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *ICRA*, 2011.

[16] A. Sharma, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Autonomous reinforcement learning via subgoal curricula," in *NeurIPS*, 2021.

[17] I. Uchendu *et al.*, "Jump-start reinforcement learning," in *ICML*, 2023.

[18] M. Dalal, D. Pathak, and R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," in *NeurIPS*, 2021.

[19] S. Nasiriany, H. Liu, and Y. Zhu, "Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks," in *ICRA*, 2022.

[20] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," in *IROS*, 2023.

[21] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "Smart-llm: Smart multi-agent robot task planning using large language models," in *IROS*, 2024.

[22] R. S. Sutton, "Reinforcement learning: An introduction," *A Bradford Book*, 2018.

[23] Y. Zeng, Y. Mu, and L. Shao, "Learning reward for robot skills using large language models via self-alignment," *arXiv*, 2024.

[24] W. Huang *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv*, 2022.

[25] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *ICML*, 2022.

[26] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning," in *CoRL*, 2023.

[27] M. Ahn *et al.*, "Do as i can and not as i say: Grounding language in robotic affordances," in *arXiv*, 2022.

[28] B. Liu *et al.*, "Llm+ p: Empowering large language models with optimal planning proficiency," in *arXiv*, 2023.

[29] F. Joublin *et al.*, "Copal: Corrective planning of robot actions with large language models," in *ICRA*, 2024.

[30] I. Singh *et al.*, "Progprompt: Generating situated robot task plans using large language models," in *ICRA*, 2023.

[31] J. Liang *et al.*, "Code as policies: Language model programs for embodied control," in *arXiv*, 2022.

[32] S. Li *et al.*, "Pre-trained language models for interactive decision-making," *NeurIPS*, 2022.

[33] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," in *AAAI*, 2024.

[34] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *IJRR*, 2013.

[35] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *JAIR*, 2018.

[36] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," in *CoRL*, 2022.

[37] Z. Zhao *et al.*, "A survey of optimization-based task and motion planning: From classical to learning approaches," *arXiv*, 2024.

[38] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *JAIR*, 2003.

[39] T. Silver *et al.*, "Predicate invention for bilevel planning," in *AAAI*, 2023.

[40] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning," in *IROS*, 2022.

APPENDIX

A. Pseudocode for Continual Skill Learning

We show the Pseudocode for the overall pipeline:

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:
$\mathcal{O}, x^{(0)}, s^{(0)} \leftarrow$ env.get_state()
$x_{\Psi}^{(0)} \leftarrow \text{PARSE}(x^{(0)}, \mathcal{O}, \bar{\Psi})$ ▷ continuous state to symbolic state
$\underline{\Omega} \leftarrow \text{GROUND}(\mathcal{O}, \bar{\Omega})$ ▷ 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

B. Prompts for Operator Creation

We show a prompt example for operation creation:

```
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:
...
```

C. Prompts for Reward Generation

We show a prompt example for reward generation:

```
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 is tasked with generating dense rewards $\mathcal{R}_{\underline{\omega}}^{D}$ by selecting relevant metric functions $\mathcal{F}_{\underline{\omega}} = \{f_1, f_2, ..., f_n\} \subseteq \mathcal{F}$ and assigning appropriate weights $\mathcal{V}_{\underline{\omega}} = \{v_1, v_2, ..., v_n\}$, s.t. $\sum_{i=1}^{n} v_i = 1$ to indicate their significance to the reward. The dense reward is therefore: $\mathcal{R}_{\underline{\omega}}^{D}(x^{(t)}) = \sum_i v_i \cdot f_i(x^{(t)}, \mathcal{O}_{\underline{\omega}})$, where $\mathcal{O}_{\underline{\omega}}$ defines a group of object entities relevant to operator $\underline{\omega}$.

D. Baseline Details for LEAGUE Tasks

To assess performance, we compare LG-SAIL against strong baselines for learning long-horizon tasks in "StackAtTarget", "StowHammer", and "PegInHole" which are **RL(SAC)**, **Curriculum RL (CRL)**, **Hierarchical RL (HRL)**, and **LEAGUE**:

- **RL (SAC)** - We utilize Soft Actor-Critic (SAC) [11] for vanilla reinforcement learning baseline;
- **Curriculum RL (CRL)** - This baseline is adopted from curriculum RL strategies [16, 17], which starts the training with near-success initializations and gradually move the reset states back to the initial task states;
- **Hierarchical RL (HRL)** - This baseline utilizes recent HRL frameworks [18, 19], which trains a high-level meta controller to compose parameterized skill primitives and atomic actions;
- **LEAGUE** [4] - This baseline reflects the recent trend of integrating TAMP frameworks with skill learning. We utilize it for this experiment due to its superior performance on long-horizon tasks.

E. Validating Skill Generalization and Adaptation

We conduct experiments to validate that our method can solve novel task goals within the domain and efficiently adapt existing skills to learn tasks in new domains.

Generalizing to new goals. In addition to training goals for "PegInHole" and "StowHammer", we directly evaluate our framework on new goals. 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. I. 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.

TABLE I: **Task goal generalization.** The results for directly testing our framework on new task goals in "StowHammer" and "PegInHole" domains without additional learning.

	Training Goal	Test Goal1	Test Goal2
"StowHammer"	0.96 ± 0.15	0.91 ± 0.09	0.71 ± 0.28
"PegInHole"	0.92 ± 0.11	0.59 ± 0.14	0.93 ± 0.06

Adapting to new objects and scenes. We aim to evaluate LG-SAIL's ability to adapt learned skills to novel situations using the LIBERO [12] 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. 6. 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.

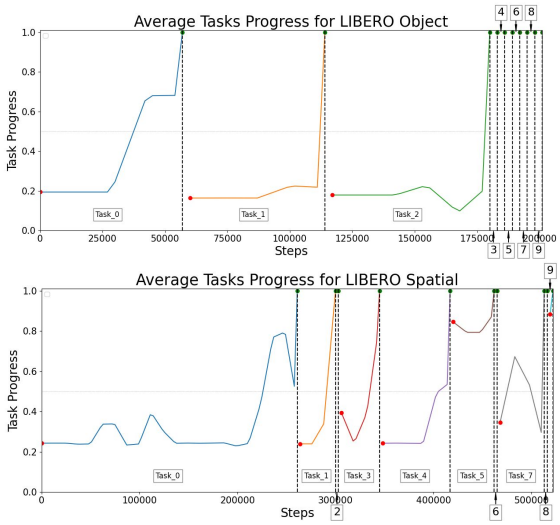

Fig. 6: **Continuous skill adaptation and learning.** The learning process is shown for both "LIBERO-Object" and "LIBERO-Spatial" tasks.

Adapting to 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 OpenCabinet and CloseCabinet from "StowHammer" in the skill library by using semantic embedding and uses these models to initialize skill policies. This strategy demonstrates improved efficiency compared to learning the task from scratch as shown in Fig. 7.

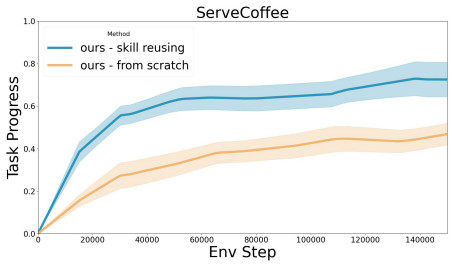

Fig. 7: **Skill adaptation in new domains.** For "MakeCoffee", we compare learning the task from scratch and learning by retrieving and adapting the skills learned from "StowHammer" domain.

TABLE II: **Task Planning Comparison.** We evaluate the performance of different designs on "StowHammer". The task planning success rate is calculated based on trials where the skills are correctly generated.

	LLM+A∗	**LLM**	**LLM 1-shot**
Skill Generation	**61.0%**	59.3%	52%
Task Planning	**100%**	78.1%	69.2%
Full Design	**100%**	**100%**	36%
Average Attempts	**1.64**	2.16	N/A

F. 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:

- A∗ Planner (**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 Planner (**LLM**): Similar to prior works on LLM-based planning [20, 21], 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 generating valid plans given correct skills, (3) overall success rate per design choice, and (4) average number of attempts. Each variant runs until 25 correct plans are generated.

Results are in Tab. II. 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 MI**) that provides only metric function's header to the LLM. Both are tested on "StowHammer" over 25 trials, evaluating reward validity for each skill. Following Xie et al. [7], 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

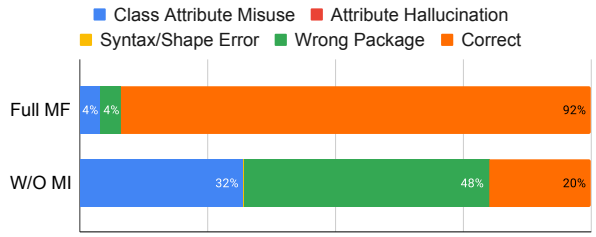

Fig. 8: **Error breakdown.** We show the error distribution of reward generation for "`StowHammer`".

TABLE III: **Reward generation comparison.** We compare reward generation success rates for different skills and the full task on "`StowHammer`".

	Full Task	Pick	Place	Open	Close
W/O MI	20%	40%	36%	44%	48%
Full MF	**92%**	**96%**	**96%**	**100%**	**100%**

on error-free reward generation, not its impact on policy optimization.

Tab. III 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. Fig. 8 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.

Related Work

G. LLMs for Reward Design

Designing effective reward functions has long been a challenge in reinforcement learning [22]. Recent studies have investigated using LLMs to generate reward functions [7–9, 14, 23]. 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 [8] 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. [23] 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.

H. LLMs for Planning and Decision Making

In recent years, LLMs have been widely explored for robot planning [24–29] and decision-making [30–32]. These studies demonstrate LLMs' ability to understand complex commands, create task plans, and translate natural language into executable actions. For instance, SayCan [27] uses LLMs to generate task plans, grounding each skill with learned control policies, while ProgPrompt [30] prompts LLMs to generate executable programs that invoke function calls for robot tasks. Researchers have also explored using LLMs to solve PDDL planning problems [33], 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.

I. TAMP and Learning for TAMP

Task and Motion Planning (TAMP) [3, 34] offers a robust framework for tackling long-horizon tasks in structured environment settings. In detail, TAMP breaks down complex planning problems into a sequence of symbolic-continuous subtasks, which simplifies the optimization for finding solutions. However, the successful deployment of TAMP systems require well-defined planning domains, which depend heavily on extensive expert knowledge. To address these limitations, recent efforts have been made to combine TAMP with learned models [5, 35–37]. To reduce reliance on manually engineered low-level skill controllers, researchers have explored learning low-level skill policies. Mandlekar et al. [6] 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. However, these approaches still rely on manually designed symbolic planners for task decomposition. In this work, we utilize LLMs to integrate TAMP planning with skill learning, enabling robots to tackle long-horizon tasks more efficiently and with minimal human intervention—a step toward continuous and lifelong robot learning.

Background and Problem Setting

Planning domain and skill representations. We focus on deterministic, fully observed tasks that can be described in PDDL [38], 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: $x(o) \in \mathbb{R}^{\texttt{dim(type}(o))}$. Predicates $\bar{\Psi}$ describe the relationships among objects in the environment. Each predicate $\bar{\psi}$ (e.g., `Holding(?object:peg)`) is characterized by a tuple of object types $(\lambda_1, ..., \lambda_m)$ and a binary classifier that

determines whether the relationship holds: $c_{\bar{\psi}} : \mathcal{X} \times \mathcal{O}^m \rightarrow \{True, False\}$, 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 $\mathcal{F}_{\bar{\psi}} = \{f_i\}$ related to this predicate: $c_{\bar{\psi}} \triangleq \bigwedge_{f_i \in \mathcal{F}_{\bar{\psi}}} \mathbb{1}[f_i \leq \epsilon_i]$, where each metric function f_i (e.g., `distance`) outputs a real number to quantify the relationships among the query objects. 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 $\bar{\Psi}$ 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 is 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: $J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t \mathcal{R}(x^{(t)}) \right]$.

Problem setting. Our setting is inspired by prior work on operator invention for bi-level planning [39, 40]. However, we assume only a small set of predefined predicates and metric functions. For each task goal g, our focus is on automating operator discovery for long-horizon task decomposition and learning primitive manipulation skills to accomplish subgoals induced the corresponding operators. In our setting, each lifted operator $\bar{\omega}$ will have a corresponding skill policy π to be learned, while during execution the ground operators belong to the same lifted operator $\bar{\omega}$ share the same skill policy.