LEAGUE++: EMPOWERING CONTINUAL ROBOT LEARNING THROUGH GUIDED SKILL ACQUISITION WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

To support daily human tasks, robots need to tackle intricate, long-term 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 tasks. 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 address these challenges, we developed LEAGUE++, a framework that leverages Large Language Models (LLMs) to harmoniously 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, LEAGUE++ maintains a symbolic skill library and utilizes the existing model from semantic-related skill to warm start the training. Our method, LEAGUE++, demonstrates superior performance compared to baselines across four challenging simulated task domains. Furthermore, we demonstrate the ability to reuse learned skills to expedite learning in new task domains.

1 INTRODUCTION

For robots to aid in daily human tasks, they need to tackle intricate long-term challenges and adapt to unfamiliar situations. While modern deep reinforcement learning (RL) techniques are promising for complete autonomous learning, they often fall short in achieving extended objectives in vast settings. Conversely, Task and Motion Planning (TAMP) methods are adept at addressing and adapting to long-term tasks due to their robust state and action abstracts. However, their reliance on predetermined skills restricts their practical use in real-world scenarios.

In an effort to overcome the constraints of both TAMP and RL, LEAGUE Cheng & Xu (2023) presents a comprehensive integration of the two methodologies. The core aim of LEAGUE is to equip robots with the proficiency to manage complex long-horizon tasks and to reuse their acquired skills in a wide range of situations. Nevertheless, the present methodology necessitates substantial prior knowledge input from humans. Specifically, humans are required to delineate certain symbolic operators for task planning and establish reward functions for the RL training process. While this approach has shown promise in enabling continuous skill acquisition efficiently, the time-intensive nature of designing symbolic operators and reward functions, along with the task-specific manual crafting of these functions, presents significant limitations. These constraints hinder the scalability of skill learning, particularly in real-world scenarios where robots regularly encounter new challenges, necessitating the continuous and automatic acquisition of skills through lifelong learning.

Recent studies, including those by Ahn et al. (2022); Singh et al. (2022); Driess et al. (2023); Wang et al. (2023); Ma et al. (2023), have sought to utilize Large Language Models (LLMs) to reduce human effort in the development of task designs and reward functions, aiming to achieve partial automation within their frameworks. However, the majority of these studies have employed an offline, one-shot approach, which necessitates the foundational model to generate reward function codes in a freeform manner. These methods heavily rely on the syntactic and semantic correctness of the outputs from large language models (LLMs). Unfortunately, generating content freely from LLMs is frequently prone to inaccuracies Huang et al. (2023a), and these papers do not consider acquiring skills in the context of long-horizon tasks. Moreover, the offline nature of these methods hampers the potential for achieving continuous skill learning within their frameworks. Given these constraints, using Large Language Models (LLMs) for code generation in lifelong learning becomes challenging.

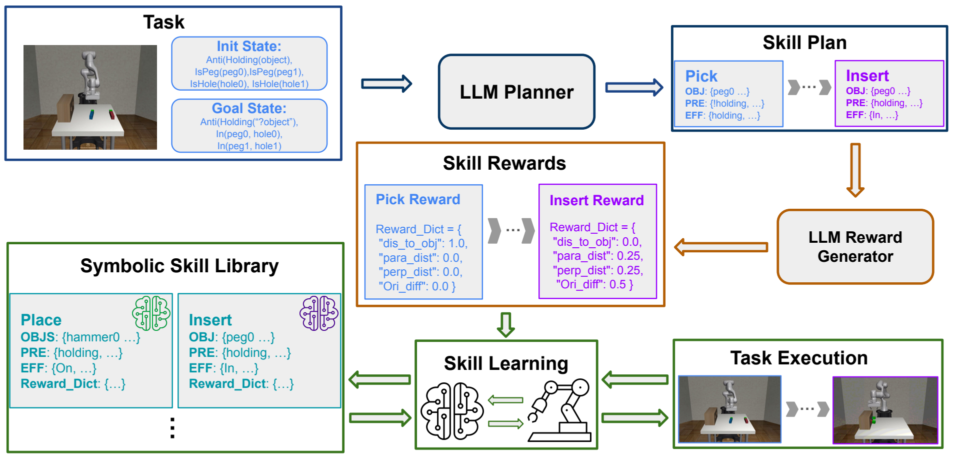

Figure 1: **The overall framework of the LEAGUE++.** We present a framework that utilizes LLMs to guide continual learning. We integrated LLMs to handle task decomposition and operator creation for TAMP, and generate dense rewards for RL skill learning, which can achieve online autonomous learning for long-horizon tasks. We also use a semantic skills library to enhance learning efficiency for new skills.

To overcome the limitations of previous works, we introduce LEAGUE++ (depicted in Fig. 1). This enhanced framework combines the strengths of LLMs for task decomposition, TAMP for defining skill boundaries and abstract environment state, and RL for skill learning, thereby overcoming the limitations of previous approaches. LEAGUE++ automates the breakdown of complex tasks by leveraging TAMP for abstracting the environment's state and defining skill limits, and then utilizing LLMs for detailed planning within those limits, significantly enhancing skills' accuracy and reusability. Each skill outlined in the plan is utilized as curriculum during training sessions, with an LLM-based reward generator providing dense rewards for each to boost efficiency. Furthermore, to prevent hallucinations by LLMs, LEAGUE++ employs metric functions for reward generation. This approach simplifies a complex coding challenge into an easily manageable selection task, thereby increasing generation accuracy. Lastly, LEAGUE++ features a symbolic skill library designed for efficient skill retrieval, facilitating skill reuse during training. With these advancements, we enhance the system's capability to continually expand its skill set across different problem domains.

With comprehensive experiments, LEAGUE++ has shown to outperform its predecessor LEAGUE and other leading RL-based methods in four intricate table-top manipulation tasks. Our ablation studies further validate LEAGUE++'s capability to generate rewards for long-horizon tasks effectively, showcasing its reliability for continuous learning and its potential for lifelong learning. To summarize, our key contributions include: 1) employing LLMs to facilitate continuous learning through automating task planning, skills creating, and dense reward generation; 2) Refining LLMs' generation by giving them structured optional context and limiting their responses rather than freeform code generation; 3) Improving the acquisition of new skills by utilizing a symbolic skills library to reuse learned skills during training. With these advancements in League++, we have significantly increased the framework's scalability across various problem domains, enabling continuous skill learning in long-horizon tasks.

2 Related Work

2.1 Language Guided Reward Design

In RL, the concept of LLM-guided reward design has emerged as a key method, utilizing language instructions and LLMs to boost agent performance and efficiency. Early efforts Goyal et al. (2019a;b) showcased the potential of using natural language to guide reward shaping, improving task completion rates in complex environments like the Atari Learning Environment. Then, the advent of frameworks like Yu et al. (2023); Anonymous (2024); Ma et al. (2023) marked a pivotal shift towards automating the generation of reward function codes using LLMs which outperform expert-crafted rewards in various domains. However, LLM-based reward generators may not suit long-horizon tasks and lifelong learning due to potential syntax errors and irrelevant content creation. Methods like EUREKA Ma et al. (2023) require a significant amount of time to develop rewards for individual skills. This process becomes increasingly prone to errors as tasks grow more complex. To solve these limitations, we propose to use LLMs to select metric functions from a human-designed functions library, streamlining reward function creation and reducing errors for these complex problem domains.

2.2 LLMs for Planning and Decision Making

LLMs have significantly transformed how robots plan and make decisions, driven by key research Ahn et al. (2022); Rana et al. (2023); Singh et al. (2022); Huang et al. (2023b). These studies demonstrate LLMs' ability to understand intricate commands, create action plans, and convert natural language into executable actions, thereby improving robot autonomy. Innovations such as dynamic replanning and corrective strategies Joublin et al. (2023), coordination in collaborative tasks Chen et al. (2023), and the merger of perception with language models for instant task modifications Skreta et al. (2024) underscore LLMs' transformative effect on robotics. Our research aims to leverage LLMs to generate the necessary skills and plans for each task, advancing the creativity of LLMs to enable the scalability of our framework across problem domains.

2.3 TAMP and Learning for TAMP

Task and Motion Planning (TAMP) Kaelbling & Lozano-Pérez (2011; 2013); Garrett et al. (2021) offers a robust framework for tackling intricate manipulation activities across broad time frames. In detail, TAMP breaks down complex planning challenges into a sequence of symbolic-continuous subtasks, which simplifies the problem-solving process. However, the successful deployment of TAMP solutions requires skills that necessitate extensive expert domain knowledge in design. To address these limitations, recent efforts have been made to combine TAMP with involved models acquired by identifying skill preconditions and outcomes Konidaris et al. (2018); Liang et al. (2022); Silver et al. (2022). For example, Konidaris et al. (2018) generates symbolic models via experimental iterations, whereas Liang et al. (2022) utilizes graph neural networks for understanding skill impacts. However, such methods still depend on the manual creation of detailed skill sets designed to accomplish the task. To address these problems, the methodologies proposed by Silver et al. (2022) and Cheng & Xu (2023) enhance TAMP systems through learned skill policies and also leverage TAMP system for state action abstraction. This synthesis allows TAMP to serve as a curriculum guiding DRL learning and simultaneously incorporates DRL-learned skills as atomic components within TAMP's planning process which significantly improves its ability to tackle contextually complex tasks.

2.4 Curriculum for RL

Curriculum Learning significantly enhances Reinforcement Learning (RL) by systematically assisting agents in skill acquisition, aiming for the mastery of complex final goals Narvekar et al. (2020). Previous research, such as Fang et al. (2021); Dong et al. (2021); Asselmeier et al. (2023), designs various curricula as different environment setups to improve model robustness. Other studies, like Sharma et al. (2021); Uchendu et al. (2023), focus on identifying specific subgoals. For instance, JumpStart Uchendu et al. (2023) employs a strategy that leverages pre-learned skills to accelerate the mastery of new tasks. Approaches such as Cheng & Xu (2023) utilize task planners to break down long-horizon tasks into atomic tasks, with each atomic task serving as a learning curriculum. Similar to Cheng & Xu (2023), our method utilizes a task planner to decompose long-horizon tasks into skill curricula, enabling their flexible combination and potential reuse in future curricula.

3 Methods

To address the need for excessive domain-specific knowledge in creating symbolic operators and dense rewards, League++ employs LLMs' innate common sense to generate solutions with respect to different problem domains. At the same time, to enhance the LLMs generator's output for planning and dense reward generation, League++ incorporates structure information of the TAMP to constrain the generation by providing a closed-set context. Furthermore, to improve skill reusability and skill learning efficiency across different problem domains, League++ maintains a library of all learned symbolic skills to warm-start the training. With advancements in League++, we have enhanced the scalability of the framework for continuous skill learning in long-horizon tasks. We present the technical details in the following sections. The diagram of our framework is shown in Fig. 1

3.1 Background

TAMP. To regularize the generation of LLMs models for both plans and rewards, we have adopted the symbolic planning interface $\langle O, \Lambda, \Psi, \Omega \rangle$ from Task and Motion Planning (TAMP). Each object $o \in O$ within the environment (for example, `hole1`), possesses a specific type $\lambda \in \Lambda$ (such as `hole`) and a tuple of dim(λ)-dimensional features containing information such as pose and size. The symbolic definition of each skill restricts the state representation to be skill-related: $\hat{x} \in \mathbb{R}^{\text{dim}(\text{type}(o))}$. Predicate $\psi \in \Psi$ defines a propositional spatial relationship between objects and between objects and the robot. A predicate ψ, such as `In(?object:peg,?object:hole)`, `holding(?object:peg)`, is defined by a tuple of object types $(\lambda_1, \ldots, \lambda_m)$ and a binary classifier $c_\psi : X \times O^m \to \{True, False\}$. This classifier determines whether the relationship holds, with each substitute entity $o_i \in O$ constrained to have a type $\lambda_i \in \Lambda$.

MDP. The motion of any generated symbolic skills can be represented as a Markov Decision Process (MDP) denoted by $\langle X, A, R(x,a), T(x'|x,a), p(x^{(0)}), \gamma \rangle$. With Continuous State Space X, Continuous Action Space A, Reward Function R, Environment Transition Model T, Distribution of Initial States $p(x^{(0)})$, Discount Factor γ. The objective for DRL training is to maximize the expected total reward J of the policy $\pi(a|x)$ that the agent employs to interact with the environment: $J = \mathbb{E}_{x^{(0)},x^{(1)},\ldots,x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t R(x^{(t)}) \right]$.

3.2 LLM-based Task Decomposition and Skill Creation

To reduce the domain knowledge required to design valid skills for TAMP, we propose leveraging the web-scale, rich semantic knowledge from LLMs to decompose the task and create reusable, atomic skills. LLMs have proven to excel at task understanding and semantic reasoning, as explored in previous work Huang et al. (2022a;b). However, they face challenges with hallucination - their generated task plans often disregard the constraints between adjacent skills and may "hallucinate" impossible action effects, leading to non-executable task plans.

To address these issues, we propose utilizing readily available structural information from the TAMP system. Specifically, the LLMs planner starts by receiving the objects $o \in O$, the initial state $\psi_{\text{init}} \in \Psi$ and the end goal $\psi_{\text{goal}} \in \Psi$ of the task, and is designed to generate a sequence of atomic symbolic skills $\omega \in \Omega_{\text{LLM}}$ specifically tailored to achieve the task's end goals from the initial state. Each symbolic skill is defined by the tuple $\langle$`Obj`, `Pre`, `Eff`$\rangle$, where `Obj` refers to the object with which the skill interacts, `Pre` denotes the precondition specifying the minimum conditions necessary for the skill's execution, and `Eff` represents the set of predicates describing the skill's objective and the expected effects resulting from the successful execution of the skill.

To enhance the accuracy of planning, we have incorporated an A* plan checker to confirm the correctness of the symbolic skill plan output. This method detects two types of errors: either the current set of symbolic skills is insufficient for achieving the task's end goal, or the order of the symbolic skills is incorrect. If the verification process fails, corresponding error feedback is given to the LLMs planner for the regeneration of the plan.

Algorithm 1 LLM Planning Algorithm

```
1: Input:
2: $\psi_{\text{init}}$                                   ▷ Task Initial State
3: $\psi_{\text{goal}}$                                   ▷ Task End Goal
4: $\Psi$                                          ▷ Set of all predicates
5: Start:
6: $\bar{\Omega}_{\text{LLM}}$, is_valid, error_feedback $\leftarrow$ [], $False$, $None$
7:                          ▷ Initialize LLM plan and error feedback
8: while not is_valid do
9:     $\bar{\Omega}_{\text{LLM}}$ = LLM_Planner($\psi_{\text{init}}, \psi_{\text{goal}}, \Psi$, error_feedback)
10:    is_valid, error_feedback = A*_planner_checker($\bar{\Omega}_{\text{LLM}}$)
11: end while
12: return $\bar{\Omega}_{\text{LLM}}$
```

As such, our LLMs planner relaxes the need for expert domain knowledge by leveraging the innovative capabilities of LLMs. At the same time, it provides a closed-set context to the LLMs and guides its generation with the logical and physical constraints of the environment by utilizing the structured information defined in the TAMP. Additionally, by integrating an A* plan checker to ensure the correctness of the final plan, we significantly minimize the chances of our LLMs planner producing an incorrect plan.

3.3 LLM-BASED REWARD GENERATION

Dense rewards are crucial for reinforcement learning, providing immediate feedback to shape an agent's behavior. However, traditional human-crafted dense rewards are typically tailored to specific problems and require considerable effort, presenting challenges for continuous and lifelong learning scenarios where agents are constantly exposed to diverse new tasks. To solve this problem, some LLM-based reward generators have been introduced Anonymous (2024); Ma et al. (2023). However, LLM-based generation methods like Eureka Ma et al. (2023) suffer from inefficiency, as it utilize evolutionary search that samples several independent outputs from the LLMs, which demands excessive in-context feedback for iteration. Moreover, these methods, primarily used for code generation, can lead to incorrect or unfeasible code, especially when complex, long-horizon tasks result in lengthy outputs. We therefore propose utilizing metric functions to tackle these challenges.

Metric Function $\mu \in M$ defines a quantitative spatial relationship between objects and between objects and the robot. These relationships are defined by a tuple of object types $(\lambda_1, \ldots, \lambda_m)$ and a continuous function $c_\psi : X \times O^m \to [0, 1]$. This normalized continuous function reflects the quantitative value of the relationship. For example, $dis\_to\_obj(?object : object)$ defines the distance between the gripper and the object. More metric functions are shown in Table. 6.2

Metric functions offer numerous advantages in our context. Instead of generating code from scratch, we simplify this problem by letting LLMs to choose metric functions and their weights to populate the dense rewards. This strategy enhances success rates of generating usable reward functions by constraining the solution space of LLMs, thus avoiding irrelevant outputs that may not pertain to the task. Additionally, the semantic information derived from the metric functions' code aids the LLMs in composing reward function that aligns well with task objectives. We assume a finite set of metric functions to describe object interactions as the common relationships and attributes of objects in daily tasks Krishna et al. (2017) can be treated as limited.

To better harness the LLMs' proficiency in code interpretation and task comprehension, we input the source code of metric functions, along with skill headers $\langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$, into the LLMs Generator. As an 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 reward generator is tasked with generating dense rewards by carefully selecting relevant metric functions $M_{\text{LLMs}} = \{\mu_1, \mu_2, ..., \mu_n\} \subseteq M$ and assigning appropriate weights $W_{\text{LLMs}} = \{w_1, w_2, ..., w_n\}$ where $\sum_{i=1}^{n} w_i = 1$ to indicate their significance to the sparse reward `Eff`.

We also define sparse rewards generated by predicates (i.e., the agent receives 1.0 when all predicates that characterize the desired effect of the skill are satisfied) to complement the dense reward constructed by LLMs. This approach facilitates more robust learning for the agent, enabling the acquisition of nuanced behaviors and the attainment of the expected effects of each skill. Finally, the task reward at any given time t is determined by:

$$R_{\text{LLMs}}(x^{(t)}) = Max[R_D, R_S]$$

where:

$$R_D = \sum_i w_i \cdot \mu_i(x^{(t)}) \quad \forall \mu_i \in M_{\text{LLMs}}, \forall w_i \in W_{\text{LLMs}}$$

$$R_S = \begin{cases} 1 & \text{if } \bigwedge_j \psi_j(x^{(t)}) \text{ for } \psi_j \in \texttt{Eff} \\ 0 & \text{otherwise} \end{cases}$$

Where R_D is dense reward, and R_S is sparse reward.

The figure of this section is shown in Fig. 5

Skill Learning. With the LLMs generated reward function, we can then optimize the policy corresponding to each symbolic skill with RL by maximizing the expected total reward: $J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t R_{\text{LLMs}}(x^{(t)}) + \alpha \mathcal{H}(\pi(\cdot | \hat{x}^{(t)})) \right]$. We adopt SAC Haarnoja et al. (2018) to optimize the skill policy, where $\mathcal{H}(\cdot)$ is the entropy term. A skill example is shown in Table. 6.1

3.4 Accelerate Learning with Symbolic Skill Library

So far, we have described how to leverage the rich semantic knowledge from LLMs to guide task decomposition and skill optimization. Another important requirement for lifelong learning is to effectively reuse existing knowledge to accelerate the learning of new tasks in new domains. Our idea is that the semantically-similar skill operators should share low-level behaviors as well. For example, opening a refrigerator and opening a door may require similar behaviors and interactions. Therefore, the policy models for existing skills should provide good initialization to warm-start the learning of new skills in novel domains.

Based on this, we propose a novel storage solution — symbolic skill library. Each element in our symbolic skill library is a pair of symbolic operator $\omega \triangleq \langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$ and a corresponding neural network weight. To utilize the skills stored in our library, for any new skills that need to be acquired, we first obtain their feature representation by extracting the LLMs embeddings of their symbolic description (i.e., $\langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$), we then identify the most similar (In our implementation, we use cosine similarity) existing skills and use its weights to initialize the new skill policy as a *warm start* for training. For every skill learned, LEAGUE++ stores its skill definition $\langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$ and its weight in the symbolic skills library. The diagram of the section can be founded in Fig. 1

4 Experiment

In this section, we aim to evaluate the performance of our framework LEAGUE++. We will compare our framework with previous baselines in four different tasks. Furthermore, we will verify the feasibility of our reward generator for enhancing continuous learning through ablation studies.

4.1 Experimental Setup

We conduct experiments in four simulated domains. We devise tasks that require multi-step reasoning, contact-rich manipulation, and long-horizon interactions. The initial state and final state of the tasks are shown in Fig. 2

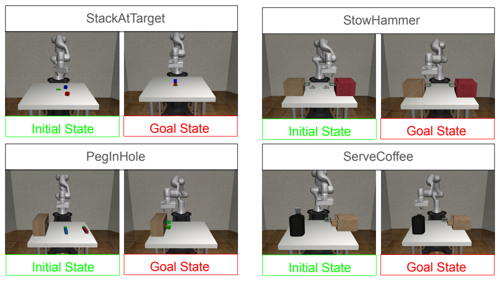

Figure 2: **Environment Setup** Four simulated domains for four tasks that require multi-step reasoning, contact-rich manipulation, and long-horizon interactions. They are StackAtTarget, StowHammer, PegInHole, and ServeHammer. .

StackAtTarget is to stack two cubes on a tight target region with a specific order. Since the cubes are randomly placed in the scene, the LLMs planner needs to make different skills to accomplish this task according to different initial states. For instance, if the second cube already occupies the target position initially, the robot must relocate it away from the target before sequentially stacking both cubes in the target area. This task serves to demonstrate the adaptability of our LLMs Planner in formulating plans that accommodate diverse initial conditions.

StowHammer is to stow two hammers into two closed cabinets. In this task, the LLMs Planner needs to make four different skills `Pick`, `Place`, `OpenCabinet`, and `CloseCabinet` and arrange them in the correct order to put the two hammers into the cabinets. Since this long-horizon task contains 8 steps, this task can well verify the accuracy of our planning system.

PegInHole is to pick up and insert two pegs into two horizontal holes. The planner needs to make two skills `Pick` and `Insert` to accomplish this task. Considering that insertion is a complex skill, this task can validate the capability of the LLM reward generator to successfully produce high-quality dense rewards, thereby completing the training process with great efficiency.

MakeCoffee is to pick up a coffee pod from a closed cabinet, insert it into the holder of the coffee machine, and finally close both the lid and the cabinet. In this experiment, we will use many skills used in previous tasks, such as `OpenCabinet`, `CloseCabinet`, `Pick`, and `Place`. This task can verify that our method can select reusable skills from the symbolic skills library and improve the training efficiency.

The environments are built on the Robosuite Zhu et al. (2022) simulator. We use a Franka Emika Panda robot arm that is controlled at 20Hz with an operational space controller (OSC), which has 5 degrees of freedom: end-effector position, yaw angle, and the position of the gripper.

4.2 Quantitative Evaluation

In this section, we aim to evaluate our framework with the previous work LEAGUE and some other SOTA baseline methods. We will introduce those baseline methods and share the quantitative evaluation results:

- **RL (SAC)**: We utilize Soft Actor-Critic (SAC) Haarnoja et al. (2018) as a robust baseline for reinforcement learning. For an equitable comparison, we enhance the basic task reward function within SAC by incorporating staged rewards, guided by an oracle task plan. This modification ensures that the reward at each step reflects the total of rewards for all completed subgoals, in addition to the reward for the ongoing subgoal. This approach is represented as *sac* in Figure 3.
- **Curriculum RL (CRL)**: We apply advanced curriculum RL strategies Sharma et al. (2021); Uchendu et al. (2023), starting with initial states near success and gradually shifting to actual starting conditions. Initial states are chosen in reverse from an oracle task plan's subgoals. We use the staged reward system mentioned earlier in SAC. This approach is labeled as *crl* in Figure 3.
- **Hierarchical RL (HRL)**: This baseline utilizes recent HRL frameworks Dalal et al. (2021); Nasiriany et al. (2022), training a meta-controller to combine skill primitives and atomic actions, based on MAPLE Nasiriany et al. (2022). This approach is labeled as *maple* in Figure 3.
- **LEAGUE**: The previous version of our framework Cheng & Xu (2023). Plans, Skill, and Reward Functions in this framework are designed and implemented by human experts, whereas those components are generated by LLMs in our framework. This approach is labeled as *league* in Fig. 3
- **Symb+RL**: An ablation baseline of LEAGUE Cheng & Xu (2023) that removes the state abstraction and retains all other features including the symbolic plan-based curriculum. This approach is labeled as *league w/o sa* in Fig. 3

For the evaluation, we adopt task progress as our metric, which is defined as the summed reward of all task stages and normalized to [0, 1]. Below we discuss the main findings based on Fig. 3.

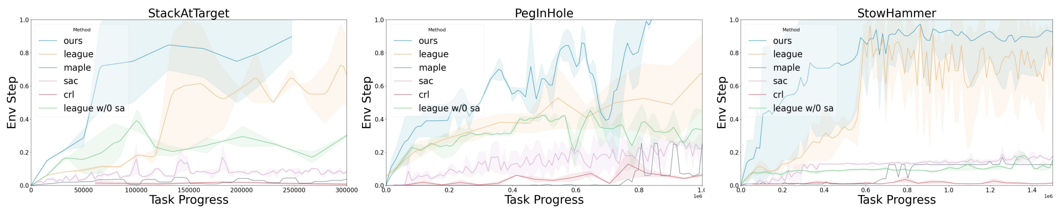

Figure 3: **Quantitative Evaluation** We compare our framework with other baselines in three task domains. The plot shows the average task progress during evaluation throughout training, which is measured as the summation of achieved rewards of each successfully executed skill in the task plan and normalized to 1. The standard deviation is shown as the shaded area.

Our framework can handle different long-horizon table-top manipulation tasks autonomously, which provides the possibility for potential lifelong learning. By inputting the environment and specific predicates of different tasks as illustrated in Figure 2, and as detailed in Sections 3.3 and 3.2, our framework autonomously executes skill generation, planning, and reward structuring. Through intensive experiments with long-horizon tasks such as StackAtTarget and StowHammer, we have validated the efficiency and accuracy of our framework's planning and skill acquisition capabilities. Our experiments reveal the framework's adeptness in dynamically adapting to various initial states in StackAtTarget, where it successfully generates appropriate skills for task completion. Similarly, in the StowHammer task, our LLMs planner demonstrated its proficiency in planning and executing a sequence of eight steps, which highlights the system's flexibility and adaptability in long-horizon tasks. Those capabilities definitely enhance the capability for continuous learning.

Our reward generator enables stable and efficient policy learning in long-horizon manipulation tasks. As shown in Fig. 3, our framework outperforms LEAGUE and other baseline methods in the longest-horizon manipulation task StowHammer, which requires 8 skills to achieve the final task goal. Since the reward of human design often cannot be combined with the TAMP structure, it will lead to a misalignment between the outcomes of learned skills and the preconditions for subsequent skills. In this case, LEAGUE will spend a lot of time retraining the previous skill after learning the later skills. Our framework can dynamically generate sparse rewards that cater to the skill effects `EFF`. Because the sparse rewards are the same as the skill effects `EFF` in the TAMP structure, this integration ensures that rewards are directly tied to the specific effects of the current skill and the initial conditions for the next skill. This strategy enhances training efficiency and efficacy compared to traditional, human-designed reward structures. This strategic advantage underscores the superior performance of our LLM-generated rewards in long-horizon manipulation tasks, which enhance the capability of continuous learning.

LLMs generated reward can outperform human-designed reward in contact-rich manipulation tasks. As shown in Fig. 3, our framework outperforms LEAGUE and other baseline methods in the contact-rich manipulation task PegInHole. Detailed in Sec. 3.3, our framework can let LLM select appropriate weights for each metric function. The advantage of using LLMs to select optimal weights for metric functions becomes particularly clear when comparing its performance to traditional human-designed weighting approaches. In the skill `Insert` in task `PegInHole`, humans might assign equal importance to metric functions such as ”`para_dis`”, ”`penp_dis`”, and ”`ori_diff`” without considering the unique demands of the task at hand. Our findings suggest that improving the weight of the metric function ”`ori_diff`” in the `PegInHole` task can improve training efficiency. The LLMs' ability to discern and apply the optimal weights for these ‘metric can enhance the performance of our framework, which showcases a significant improvement over human-devised rewards as shown in Fig. 3.

This superiority stems from the LLMs' comprehensive understanding of the task's dynamics and the intricate interactions involved in contact-rich environments. By analyzing vast amounts of data and learning from varied task executions, the LLMs can identify patterns and priorities that may not be immediately apparent to human designers. This deep, data-driven insight allows the LLMs to optimize reward functions directly aligned with the specific objectives of each task, which leads to a more effective and efficient learning process.

Symbolic skills library enhances training efficiency for new long-horizon manipulation tasks. We also set up an experiment to evaluate the feasibility of speeding up policy learning in novel domains by reusing previously acquired skills as warm-start. In the task MakeCoffee shown in Fig. 4, LLMs selected skills `OpenCabinet`, `CloseCabinet` from the StowHammer, and selected skills `Pick` and `Place` from StackAtTarget as pre-trained weights from the symbolic skill library for policy initialiazation. This strategy of reusing pre-trained skills for fine-tuning significantly enhances training efficiency, the time to reach proficiency in the MakeCoffee task was reduced a lot. As we continue to expand our symbolic skills library, each new skill added becomes a potential catalyst for more rapid training in future tasks. It supports continuous learning in robotic systems.

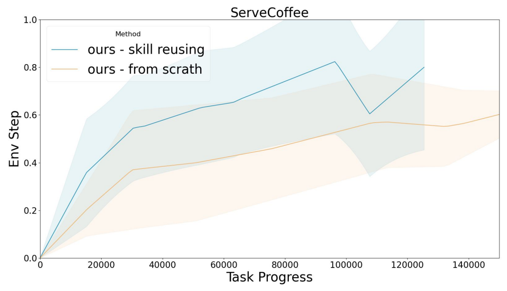

Figure 4: **Quantitative Evaluation** We compare the training efficiency of our framework with reused pre-trained skills in our symbolic library with our framework learning from scrath.

4.3 Ablation Study

In this part, we compare two design choices for using LLMs for generating dense rewards:

- **LEAGUE++** In our framework, we add the metrics inspector as a part of the input in the reward generator. It can read the code of all metric functions along with the comments and variable names in the codes.

- **LEAGUE++ w/o MI** In this ablation, we changed the input from the metrics inspector to only the metric function's header.

We use the StowHammer task to quantitatively evaluate different LLM-based reward generators, where the reward function of each skill for the tasks needs to be generated. We check whether the reward function can be executed correctly or not. Same as Anonymous (2024), we classify them into four error types: Class attributes misuse — chooses the wrong objects for the metric function; Attributes hallucination — refers to attributes that do not exist; Syntax/shape error — generates incorrect dictionary; Wrong package — selects incorrect metric functions. We test StowHammer 25 times and count the probability of each error. Also, we counted the success rate of generating reward functions for each skill `Pick`, `Place`, `Open`, and `Close`, totaling one hundred times.

Task/Skill	LEAGUE++	w/o MI
StowHammer	92%	20%
Pick	96%	40%
Place	96%	36%
Open	100%	44%
Close	100%	48%
Skill Average	98%	42%

Table 1: **Quantitative Evaluation** This table shares the success rates for correct executions of generated rewards in the task StowHammer and its corresponding skills. This table compares our proposal LEAGUE++ with the corresponding ablation LEAGUE++ without Metrics Inspector (w/o MI)

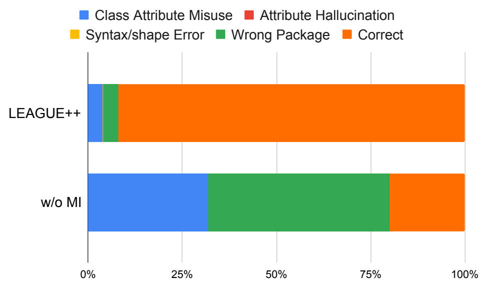

Table 2: **Error breakdown** This figure shares the error rates/success rates for reward generation with the task StowHammer. This figure compares our proposal LEAGUE++ with the corresponding ablation LEAGUE++ without Metrics Inspector (w/o MI).

According to the results shown in Table. 1, we find that our method can reach 92% success rate for generating reward functions for all skills in the task. Specifically, the success rates for individual skills—`Pick` and `Place` at 96% each, with `Open` and `Close` achieving perfect scores of 100%—culminate in an overall success rate of 98%. Therefore, we demonstrate that our method exhibits a low incidence of errors, which demonstrates the reliability of this method for long-horizon tasks and the potential of using it for lifelong learning.

As Table 2 shows, it's important to highlight that LEAGUE++ addresses two significant issues that arise LLMs generate code: the invention of nonexistent attributes and the production of syntax errors. In contrast to previous LLM reward generators Ma et al. (2023); Anonymous (2024), the problems of selecting incorrect class attributes or packages in LEAGUE++ are more likely to be resolved through improved prompts and the input of more detailed metric functions. Table 1 showcases the metrics inspector's effectiveness in drastically reducing the error rate from 80% to a mere 8% in generating incorrect class attributes and wrong packages. It signifies that the LLMs can accurately identify and select the appropriate metric functions for each skill by inspecting the mertic functions' code. These findings suggest that our current errors are likely to improve with the provision of more detailed input, offering a promising pathway towards achieving potential lifelong learning.

5 Conclusion, Limitation, and Future Work

Our paper presents LEAGUE++, a framework that synergizes LLM-guided skill learning with TAMP, enhancing RL's exploration and autonomous learning for continuous learning across diverse tasks. We demonstrate that LEAGUE++ outperforms traditional methods by using LLMs for reward generation and a semantic skills library for efficient skill acquisition. In ablation, our results also indicate high success rates in generating executable rewards, suggesting the framework's viability for lifelong learning. Although LEAGUE++ reduces the need for extensive planning and reward engineering, it inherits TAMP's assumptions regarding access to shared predicates and metric functions for domain construction and evaluation. Future work will consider advances in object relationship modeling Krishna et al. (2017) and predicate learning Migimatsu & Bohg (2022) to further mitigate these assumptions.

REFERENCES

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 J 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. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

Anonymous. Text2reward: Dense reward generation with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=tUM39YTRxH`.

Max Asselmeier, Zhaoyi Li, Kelin Yu, and Danfei Xu. Evolutionary curriculum training for drl-based navigation systems, 2023.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems?, 2023.

Shuo Cheng and Danfei Xu. League: Guided skill learning and abstraction for long-horizon manipulation. *IEEE Robotics and Automation Letters*, 8(10):6451–6458, 2023. doi: 10.1109/LRA.2023.3308061.

Murtaza Dalal, Deepak Pathak, and Ruslan Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=48uzkHOKMfz`.

Siyuan Dong, Devesh K. Jha, Diego Romeres, Sangwoon Kim, Daniel Nikovski, and Alberto Rodriguez. Tactile-rl for insertion: Generalization to objects of unknown geometry. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6437–6443, 2021. URL `https://api.semanticscholar.org/CorpusID:233004667`.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023.

Kuan Fang, Yuke Zhu, Silvio Savarese, and L. Fei-Fei. Adaptive procedural task generation for hard-exploration problems. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=8xLkv08d70T`.

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

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning, 2019a.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 2385–2391. International Joint Conferences on Artificial Intelligence Organization, 7 2019b. doi: 10.24963/ijcai.2019/331. URL `https://doi.org/10.24963/ijcai.2019/331`.

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

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023a.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022b.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models, 2023b.

Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. Copal: Corrective planning of robot actions with large language models, 2023.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

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

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123:32–73, 2017.

Jacky Liang, Mohit Sharma, Alex LaGrassa, Shivam Vats, Saumya Saxena, and Oliver Kroemer. Search-based task planning with learned skill effect models for lifelong robotic manipulation. In *ICRA*, 2022.

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

Toki Migimatsu and Jeannette Bohg. Grounding predicates through actions. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 3498–3504. IEEE, 2022.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: a framework and survey. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7477–7484, 2022. doi: 10.1109/ICRA46639.2022.9812140.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In *7th Annual Conference on Robot Learning*, 2023. URL `https://openreview.net/forum?id=wMpOMO0Ss7a`.

Archit Sharma, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Autonomous reinforcement learning via subgoal curricula. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=ELU8Bu1Z9w1`.

Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *CoRL*, 2022.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022.

Marta Skreta, Zihan Zhou, Jia Lin Yuan, Kourosh Darvish, Alán Aspuru-Guzik, and Animesh Garg. Replan: Robotic replanning with perception and language models, 2024.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning, 2023.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models, 2023.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis, 2023.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning, 2022.

6 Appendix

6.1 Skill Operator Definition

We show the symbolic definition of the `Insert` skill example.

```
Insert(?object,?hole)
    Obj: [?object:peg,?hole:hole]
    Pre: {Holding(?object),
                  IsClear(?hole)}
    Eff: {Holding(?object),
    Anti(All(IsClear(?hole))),
    Anti(All(Holding(?object))),
    In(?object,?hole)}
```

6.2 Metric Functions

We show some exemples of the Metrics Functions.

Metrics Function	Definition
dis_to_obj	Distance between the gripper and the object
$parallel\_dis$	Distance between two objects' shadows on a parallel plane
$perpendicular\_dis$	Distance between two objects along the normal line
$orientation\_diff$	Angle difference between object and hole
$open$	Distance between the handle and cabinet is large enough or not.

6.3 Reward Generation

We show the reward generation process of our method in Fig. 5.

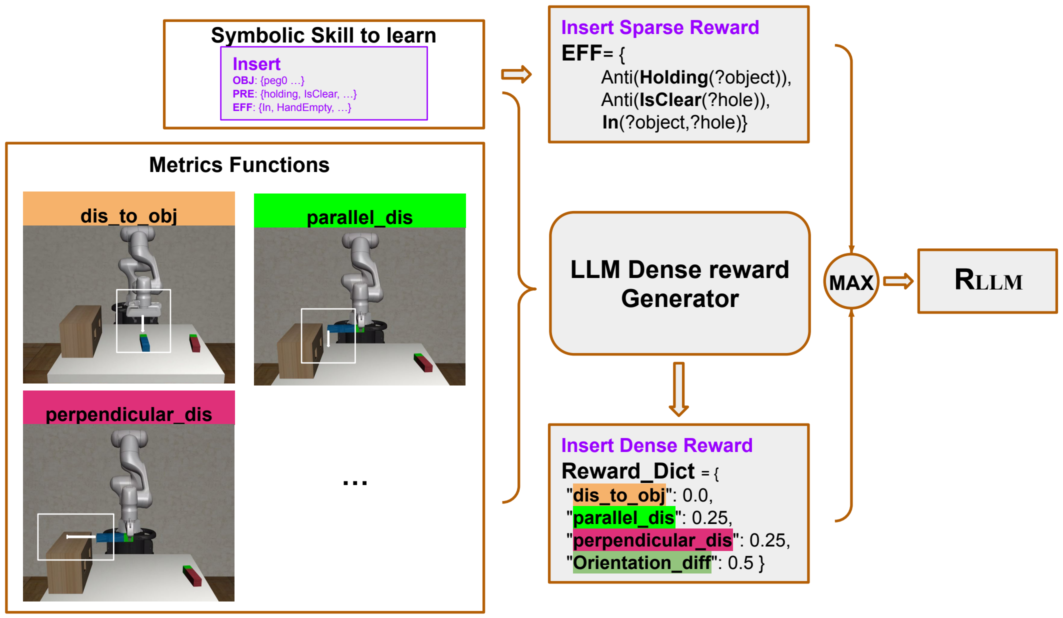

Figure 5: The reward generator takes in the skills header, outputs the dense reward for the skill.