EMPOWERING CONTINUAL ROBOT LEARNING THROUG GUIDED SKILL ACQUISITION WITH LANGUAGE MODELS

Zhaoyi Li[1*], Kelin Yu[1*], Shuo Cheng[1*], and Danfei Xu[12]

***Abstract*— To support daily human tasks, robots must handle complex, long-term tasks and continuously learn new skills. Deep reinforcement learning (DRL) offers potential for fine-grained skill learning but faces challenges with long-horizon tasks and relies heavily on human-defined rewards. Task and Motion Planning (TAMP) excel at long-horizon tasks but require tailored domain-specific skills, limiting practicality. To address these challenges, we developed LEAGUE++, integrating Large Language Models (LLMs) with TAMP and DRL for continuous skill learning. Our framework automates task decomposition, operator creation, and dense reward generation for efficient skill acquisition. LEAGUE++ maintains a symbolic skill library and utilizes existing models for warm-starting training to facilitate new skill learning. Our method outperforms baselines across four challenging simulated task domains and demonstrates skill reuse to expedite learning in new domains. Video results available at: `https://sites.google.com/view/continuallearning`.**

I. Introduction

Robots need to handle complex, long-term tasks and adapt to new situations to assist humans daily. While modern deep reinforcement learning (RL) shows promise for autonomous learning, it struggles with long-horizon objectives in diverse settings. On the other hand, Task and Motion Planning (TAMP) methods excel at addressing long-term tasks but rely on predetermined skills, limiting real-world applicability.

LEAGUE integrates TAMP and RL to enable robots to manage complex tasks and reuse skills across situations [1]. However, it requires significant human input to define symbolic operators and reward functions, hindering scalability in real-world scenarios.

Recent studies aim to reduce human effort in task design and reward function development using Large Language Models (LLMs) [2–6]. However, these methods often rely on offline, one-shot approaches and overlook continuous skill learning for long-horizon tasks.

We introduce LEAGUE++, a framework integrating LLMs, TAMP, and RL to overcome previous limitations. LEAGUE++ automates task breakdown, enhances skill accuracy and reusability, and composes metric functions for constructing dense rewards to prevent LLM hallucinations. It features a symbolic skill library for efficient skill retrieval and expands skill sets across problem domains. In experiments, LEAGUE++ outperforms its predecessor and other RL-based methods in table-top manipulation tasks, showcasing its reliability and potential for lifelong learning. Key contributions include LLMs for task planning and reward generation, structured optional context for LLM generation, and a symbolic skills library for skill reuse. These advancements significantly increase scalability across problem domains and enable continuous skill learning in long-horizon tasks.

*Equally Contributed

[1]School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA

[2]NVIDIA Corporation, Santa Clara, CA, USA

II. Methods

LEAGUE++ integrates Large Language Models (LLMs) with TAMP and DRL for continuous skill learning. We explain how to utilize LLM for automatic task decomposition and skill operator creation in Sec. II-A; we explain how to accelerate skill policy learning with LLM-based reward generator in Sec. II-B; and we present the details of how to maintain a skill library to facilitate the learning of new skills in new problem settings in Sec. II-C. The diagram of our framework is shown in Fig. 1. Due to page limit, please refer Sec. B in appendix for more background information.

A. 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 [7, 8]. 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 $\langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$, 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

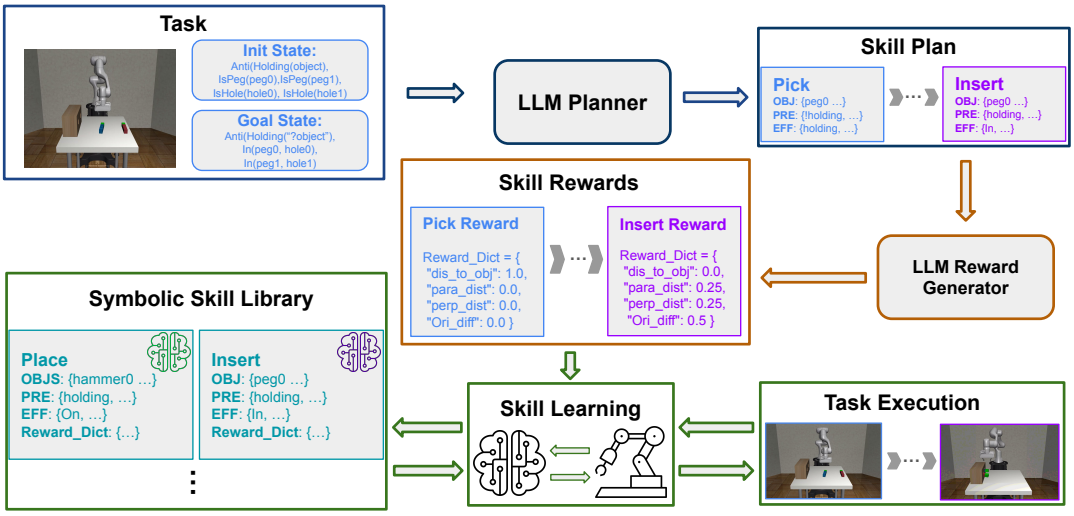

Fig. 1: **The overall framework of the LEAGUE++.** We present a framework that utilizes LLMs to guide continual learning. We integrated LLMs to handle task decomposition and operator creation for TAMP, and generate dense rewards for RL skill learning, which can achieve online autonomous learning for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

fails, corresponding error feedback is given to the LLMs planner for the regeneration of the plan. Pseducode is shown in Alg. 1

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.

B. 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 [6, 9]. However, LLM-based generation methods like Eureka [6] 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 $c_\psi : X \times O^m \rightarrow [0,1]$. 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. E

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 [10] 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 a dense reward, and R_S is a sparse reward. The figure of this section is shown in Fig. 4

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: $J = \mathbb{E}_{x^{(0)},x^{(1)},\ldots,x^{(H)}\sim\pi,p(x^{(0)})}\left[\sum_t \gamma^t R_{\text{LLMs}}(x^{(t)}) + \alpha\mathscr{H}(\pi(\cdot|\hat{x}^{(t)}))\right]$. We adopt SAC [11] to optimize the skill policy, where $\mathscr{H}(\cdot)$ is the entropy term. A skill example is shown in Table. C

C. 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

III. Experiment

To validate the efficacy of our method, we developed 4 challenging task domains with Robosuite [12] simulator, more details about each domain can be found in Sec. G. We quantitatively evaluate LEAGUE++ and other baselines on learning to solve long-horizon tasks in Sec. III-A, we then validate the effectiveness of reusing learned skills for learning new tasks. We compare and analyze different design choices for LLM-based reward generation in Sec. III-B.

A. Quantitative Evaluation

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

- **RL (SAC):** We utilize Soft Actor-Critic (SAC) [11] 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 Fig. 2.
- **Curriculum RL (CRL):** We apply advanced curriculum RL strategies [13, 14], 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 Fig. 2.
- **Hierarchical RL (HRL):** This baseline utilizes recent HRL frameworks [15, 16], training a meta-controller to combine skill primitives and atomic actions, based on MAPLE [16]. This approach is labeled as *maple* in Fig. 2.
- **LEAGUE**: The previous version of our framework [1]. 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. 2
- **Symb+RL:** An ablation baseline of LEAGUE [1] 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. 2

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. 2.

Our framework can handle different long-horizon table-top manipulation tasks autonomously, which provides the possibility for potential lifelong learning. Our framework autonomously executes skill generation, planning, and reward structuring, by leveraging the environment abstractions of each task as shown in Fig. 5. 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 8 skills, which highlights the system's flexibility and adaptability in long-horizon

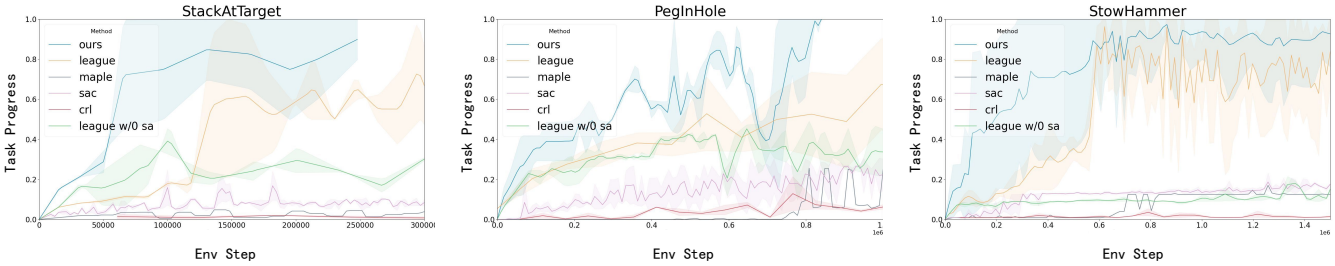

Fig. 2: **Quantitative Evaluation.** We compare our framework with other baselines in three task domains. The plot shows the average task progress during evaluation throughout training, which is measured as the summation of achieved rewards of each successfully executed skill in the task plan and normalized to 1. The standard deviation is shown as the shaded area.

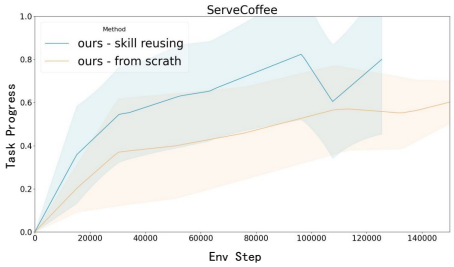

Fig. 3: **Quantitative Evaluation.** We compare the training efficiency of our framework with reused pre-trained skills with our framework learning from scratch.

tasks. Those capabilities definitely enhance the capability for continuous learning.

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. 3, 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 initialization. This strategy of reusing pre-trained skills for fine-tuning significantly enhances training efficiency, the time to reach proficiency in the ServeCoffee 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.

B. 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 [9], 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.

TABLE I: **Quantitative Evaluation.** This table shares the success rates for correct executions of generated rewards in the StowHammer task domain. This table compares our proposal LEAGUE++ with the corresponding ablated version without Metrics Inspector (w/o MI).

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%

According to the results shown in Table. I, we find that our method can reach 92% success rate for generating reward functions. 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. The error breakdown and analysis can be found in Sec. H.

References

[1] S. Cheng and D. Xu, "League: Guided skill learning and abstraction for long-horizon manipulation," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6451–6458, 2023.

[2] M. Ahn *et al.*, *Do as i can, not as i say: Grounding language in robotic affordances*, 2022. arXiv: 2204.01691 [cs.RO].

[3] I. Singh *et al.*, *Progprompt: Generating situated robot task plans using large language models*, 2022. arXiv: 2209.11302 [cs.RO].

[4] D. Driess *et al.*, *Palm-e: An embodied multimodal language model*, 2023. arXiv: 2303.03378 [cs.LG].

[5] L. Wang *et al.*, *Gensim: Generating robotic simulation tasks via large language models*, 2023. arXiv: 2310.01361 [cs.LG].

[6] Y. J. Ma *et al.*, *Eureka: Human-level reward design via coding large language models*, 2023. arXiv: 2310.12931 [cs.RO].

[7] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *arXiv preprint arXiv:2201.07207*, 2022.

[8] W. Huang *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *arXiv preprint arXiv:2207.05608*, 2022.

[9] Anonymous, "Text2reward: Dense reward generation with language models for reinforcement learning," in *The Twelfth International Conference on Learning Representations*, 2024.

[10] R. Krishna *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, pp. 32–73, 2017.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG].

[12] Y. Zhu *et al.*, *Robosuite: A modular simulation framework and benchmark for robot learning*, 2022. arXiv: 2009.12293 [cs.RO].

[13] A. Sharma, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Autonomous reinforcement learning via subgoal curricula," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.

[14] I. Uchendu *et al.*, *Jump-start reinforcement learning*, 2023. arXiv: 2204.02372 [cs.LG].

[15] M. Dalal, D. Pathak, and R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.

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

[17] P. Goyal, S. Niekum, and R. J. Mooney, *Using natural language for reward shaping in reinforcement learning*, 2019. arXiv: 1903.02020 [cs.LG].

[18] P. Goyal, S. Niekum, and R. 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*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 2385–2391.

[19] W. Yu *et al.*, *Language to rewards for robotic skill synthesis*, 2023. arXiv: 2306.08647 [cs.RO].

[20] 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 *7th Annual Conference on Robot Learning*, 2023.

[21] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, *Voxposer: Composable 3d value maps for robotic manipulation with language models*, 2023. arXiv: 2307.05973 [cs.RO].

[22] G. Wang *et al.*, *Voyager: An open-ended embodied agent with large language models*, 2023. arXiv: 2305.16291 [cs.AI].

[23] H. Yuan *et al.*, *Skill reinforcement learning and planning for open-world long-horizon tasks*, 2023. arXiv: 2303.16563 [cs.LG].

[24] F. Joublin *et al.*, *Copal: Corrective planning of robot actions with large language models*, 2023. arXiv: 2310.07263 [cs.RO].

[25] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, *Scalable multi-robot collaboration with large language models: Centralized or decentralized systems?* 2023. arXiv: 2309.15943 [cs.RO].

[26] M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, and A. Garg, *Replan: Robotic replanning with perception and language models*, 2024. arXiv: 2401.04157 [cs.RO].

[27] M. Dalal, T. Chiruvolu, D. S. Chaplot, and R. Salakhutdinov, "Plan-seq-learn: Language model guided RL for solving long horizon robotics tasks," in *CoRL 2023 Workshop on Learning Effective Abstractions for Planning (LEAP)*, 2023.

[28] M. Parakh, A. Fong, A. Simeonov, T. Chen, A. Gupta, and P. Agrawal, *Lifelong robot learning with human assisted language planners*, 2023. arXiv: 2309.14321 [cs.RO].

[29] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[30] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

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

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

[33] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer, "Search-based task planning with learned skill effect models for lifelong robotic manipulation," in *ICRA*, 2022.

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

[35] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *J. Mach. Learn. Res.*, vol. 21, no. 1, Jan. 2020.

[36] K. Fang, Y. Zhu, S. Savarese, and L. Fei-Fei, "Adaptive procedural task generation for hard-exploration problems," in *International Conference on Learning Representations*, 2021.

[37] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez, "Tactile-rl for insertion: Generalization to objects of unknown geometry," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6437–6443, 2021.

[38] M. Asselmeier, Z. Li, K. Yu, and D. Xu, *Evolutionary curriculum training for drl-based navigation systems*, 2023. arXiv: 2306.08870 [cs.RO].

APPENDIX

A. Related Work

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 [17, 18] 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 [6, 9, 19] 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 [6] 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) LLMs for Planning and Decision Making: LLMs have significantly transformed how robots plan and make decisions, driven by key research [2, 3, 20–23]. 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 [24], coordination in collaborative tasks [25], and the merger of perception with language models for instant task modifications [26] underscore LLMs' transformative effect on robotics. In paper [27, 28], the authors utilize LLM to do long-horizon task planning and then move one step forward to life-long learning with humans in loop. 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 without any human efforts.

3) TAMP and Learning for TAMP: Task and Motion Planning (TAMP) [29–31] 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 [32–34]. For example, [32] generates symbolic models via experimental iterations, whereas [33] 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 [34] and [1] 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 complex tasks.

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 [35]. Previous research, such as [36–38], designs various curricula as different environment setups to improve model robustness. Other studies, like [13, 14], focus on identifying specific subgoals. For instance, JumpStart [14] employs a strategy that leverages pre-learned skills to accelerate the mastery of new tasks. Approaches such as [1] utilize task planner with manually defined symbolic operators to break down long-horizon tasks into atomic tasks, with each atomic task serving as a learning curriculum. Similar to [1], our method utilizes LLMs for creating symbolic operators to decompose long-horizon tasks into skill curricula, enabling automatic and continuous skill learning in various complex task domains.

B. 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 $\dim(\lambda)$-dimensional features containing information such as pose and size. The symbolic definition of each skill restricts the state representation to be skill-related: $\hat{x} \in \mathbb{R}^{\dim(\text{type}(o))}$. 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: $J = \mathbb{E}_{x^{(0)}, x^{(1)}, \ldots, x^{(H)} \sim \pi, p(x^{(0)})} \left[\sum_t \gamma^t R(x^{(t)}) \right]$.

C. 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)}
```

D. LLM Planner

We show the pseudocode for the LLM Planner.

Algorithm 1 LLM Planning Algorithm

Input:
ψ_{init} ▷ Task Initial State
ψ_{goal} ▷ Task End Goal
Ψ ▷ Set of all predicates
Start:
$\bar{\Omega}_{\text{LLM}}, \text{is\_valid}, \text{error\_feedback} \leftarrow [], \mathit{False}, \mathit{None}$
▷ Initialize LLM plan and error feedback
while not is_valid **do**
    $\bar{\Omega}_{\text{LLM}} = \text{LLM\_Planner}(\psi_{\text{init}}, \psi_{\text{goal}}, \Psi, \text{error\_feedback})$
    $\text{is\_valid}, \text{error\_feedback} = \text{A*\_planner\_checker}(\bar{\Omega}_{\text{LLM}})$
end while
return $\bar{\Omega}_{\text{LLM}}$

E. 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`	Objects-Objects distance shadows on a parallel plane
`perpendicular_dis`	Objects-Objects distance along the normal line
`orientation_diff`	Angle difference between object and hole
`open`	Handle-Cabinet distance is large enough or not.

F. Reward Generation

We show the reward generation process in Fig. 4.

G. 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. 5.

a) 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.

b) 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 skills, this task can well verify the accuracy of our planning system.

c) 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.

d) ServeCoffee: 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 [12] 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.

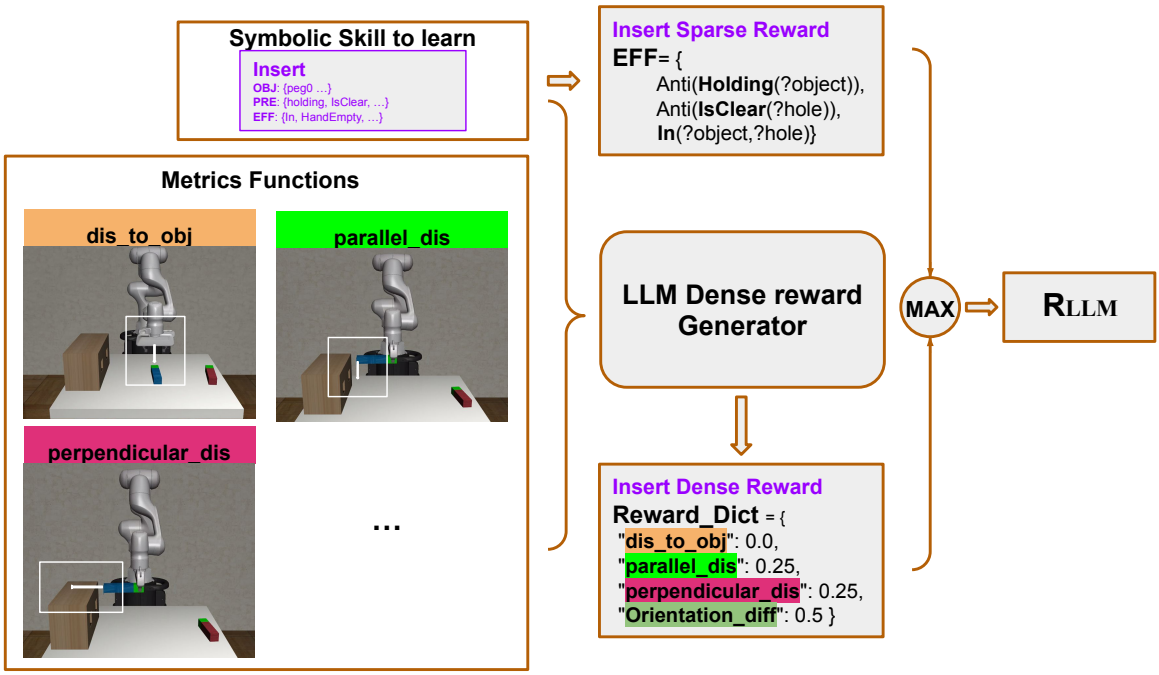

Fig. 4: **Reward Generation.** The reward generator takes in the skill definition $\langle \texttt{Obj}, \texttt{Pre}, \texttt{Eff} \rangle$, as well as all the Metrics Functions M from the Metrics Function Library, and subsequently produces a dense reward for the skill. This dense reward is then combined with the sparse reward `Eff` from the skill definition to calculate the final reward for the skill.

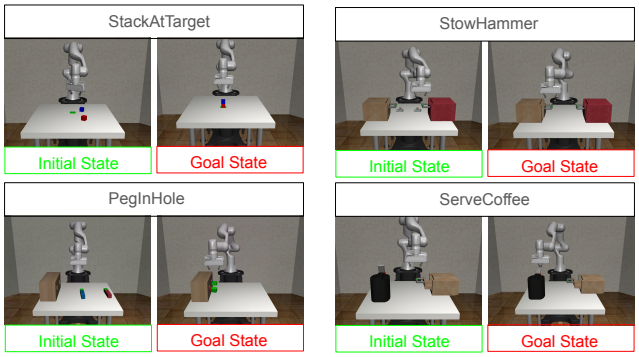

Fig. 5: **Environment Setup.** Four simulated domains for four tasks that require multi-step reasoning, contact-rich manipulation, and long-horizon interactions. They are StackAtTarget, StowHammer, PegInHole, and ServeCoffee. .

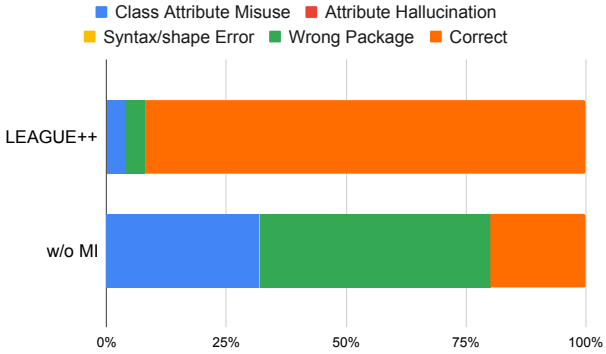

Fig. 6: **Error breakdown.** This figure shares the error rates/success rates for reward generation with the task StowHammer. This figure compares our proposal LEAGUE++ with the corresponding ablation LEAGUE++ without Metrics Inspector (w/o MI).

H. Analysis of Reward Generation Errors

As Figure 6 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 [6, 9], 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 I 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.

I. Qualitative Results

We show the task demos of our system in Fig. 7.

Fig. 7: **Task demos.** We show the demos of our experiments. We visualize the progress of each task in the Robosuite [12] environment. The first row is PegInHole; the second row is CubeReorder; the third row is HammerPlace; the last row is ServeCoffee.