2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Reward Evolution with Graph-of-Thoughts: A Bi-Level Language Model Framework for Reinforcement Learning

Changwei Yao[1], Xinzi Liu[2], Chen Li[1], and Marios Savvides[1†]
[1]Carnegie Mellon University, [2]University of Tokyo

***Abstract*— Designing effective reward functions remains a major challenge in reinforcement learning (RL), often requiring considerable human expertise and iterative refinement. Recent advances leverage Large Language Models (LLMs) for automated reward design, but these approaches are limited by hallucinations, reliance on human feedback, and challenges with handling complex, multi-step tasks. In this work, we introduce Reward Evolution with Graph-of-Thoughts (RE-GoT), a novel bi-level framework that enhances LLMs with structured graph-based reasoning and integrates Visual Language Models (VLMs) for automated rollout evaluation. RE-GoT first decomposes tasks into text-attributed graphs, enabling comprehensive analysis and reward function generation, and then iteratively refines rewards using visual feedback from VLMs without human intervention. Extensive experiments on 10 RoboGen and 4 ManiSkill2 tasks demonstrate that RE-GoT consistently outperforms existing LLM-based baselines. On RoboGen, our method improves average task success rates by 32.25%, with notable gains on complex multi-step tasks. On ManiSkill2, RE-GoT achieves an average success rate of 93.73% across four diverse manipulation tasks, significantly surpassing prior LLM-based approaches and even exceeding expert-designed rewards. Our results indicate that combining LLMs and VLMs with graph-of-thoughts reasoning provides a scalable and effective solution for autonomous reward evolution in RL.**

I. INTRODUCTION

Reinforcement learning (RL) has been successfully deployed in many fields, especially in robotics where agents learn complex skills in open environments [1]–[3]. Simultaneously, emerging simulators enhance the simulation world with more details to eliminate the sim-to-real gap and expedite the RL training process. However, one of the key challenges of applying RL is designing an appropriate reward function that will lead to the desired behavior of robots and requiring extensive tuning to optimize the efficacy, called reward engineering. To mitigate this problem, inverse RL [4], [5] and preference-based RL [6], [7] were designed in prior work. But it still requires great efforts in collecting demonstrations and suffers from limited reward diversity and too strong bias from experts, and thus is prone to overfitting and less capable of generalization.

Recently, Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning and coding, enabling them to analyze complex problems, generate efficient solutions, and even assist in debugging and optimization. Thereby, prior work has studied replacing human supervision by LLMs to write code-based reward functions [8], [9]. However, one of the major challenges with LLMs is their tendency to hallucinate, generating incorrect or misleading information with high confidence, particularly when solving complex problems. Since it occurs frequently [10] and reduces the efficacy of reward functions, some studies mitigate this limitation with human feedback in the loop, which still relies on expert knowledge. Furthermore, directly generating reward functions from LLMs is often insufficient for long-horizon, multi-step tasks due to limited structured reasoning over the complex topology of robot tasks.

To further enhance LLMs' reasoning capabilities and reduce reliance on human supervision, recent advancements such as Chain-of-Thought (CoT) [11]–[13] and Graph-of-

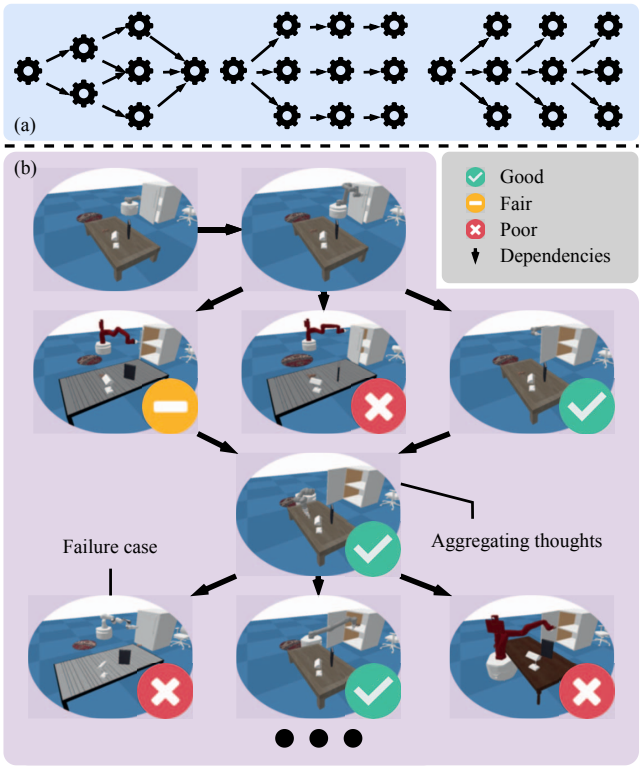

Fig. 1. Conceptual illustration of GoT. (a) Three general GoT examples, where each node is a thought. (b) GoT applied to a manipulation task named *Store Item in Storage*: nodes represent sub-goal stages, and edges represent robot behaviors for transitioning between stages, both with detailed textual descriptions.

†Corresponding author: Marios Savvides.
[1]Carnegie Mellon University, Pittsburgh, PA 15213, United States {changwey, chenli4, marioss}@andrew.cmu.edu
[2]The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8654, Japan liuxz@g.ecc.u-tokyo.ac.jp

Thought (GoT) [14], [15] have been introduced. These methods enable structured reasoning, with GoT particularly excelling in handling complex decision-making tasks by organizing information in a graph-based manner rather than a linear sequence. By leveraging GoT as shown in Fig. 1, LLMs can explore multiple reasoning paths simultaneously, making them more effective in generating well-structured and coherent reward functions. This structured approach not only mitigates hallucinations but also improves the adaptability of RL systems in dynamic environments.

In this work, we aim to eliminate human intervention in mitigating LLM hallucinations by proposing Reward Evolution with Graph-of-Thoughts (RE-GoT), a framework that enables LLMs to construct a graph-based representation of the task's solution before generating reward functions. We posit that the performance on long-horizon tasks is not merely a function of model scale, but is significantly bolstered by an architectural scaffolding that imposes necessary structural constraints to effectively organize and ground this knowledge. RE-GoT addresses this by enforcing a bi-level reasoning topology. The lower level only needs the environment codes and a text description of the task goal, mapping out a graph-based solution to the final goal first and designing the reward functions afterwards. Meanwhile, the upper level requires only a rollout video of the trained RL agent and leverages feedback from vision-language foundation models (VLMs) trained on large-scale multimodal datasets, thereby replacing human supervision, mitigating bias, and enabling a closed-loop update process. Furthermore, rather than instructing LLMs to generate the reward functions directly [8], [9], [16], equipping them with the GoT ability enables LLMs to consider more comprehensively, particularly when dealing with long-horizon, multi-step, and complex tasks with multiple solutions, making this difference significant.

In summary, our key contributions are as follows: a) we introduce the first to leverage GoT in automatic adaptive reward generation, enabling more structured and effective task decomposition, b) we propose a bi-level framework that improves reward function learning by leveraging VLMs feedback from video demonstrations without human, allowing for more adaptive policy training, and c) we demonstrate the adaptability of our approach across different platforms, showcasing its potential as a mobile tool through extensive comparative analysis and ablation studies.

II. RELATED WORK

Reward Design. Reward engineering remains a significant challenge in RL [17]–[19]. The performance of an RL agent trained via reward shaping is heavily dependent on the quality of its reward function. Many studies investigate the construction of high-quality reward functions from diverse perspectives. Inverse Reinforcement Learning (IRL) [20]–[22] was introduced to infer rewards autonomously from expert demonstrations, but it still suffers from high data acquisition costs and produces black-box reward models that lack interpretability and are difficult to adapt. Preference-based learning [7], [23], which builds reward functions by having humans compare different behaviors or trajectories, also fails to eliminate the reliance on costly annotations and introduces noise due to inconsistent individual preferences. In contrast, our RE-GoT framework requires only a few examples to adapt across tasks, effectively overcoming the high data and annotation costs of prior methods.

LLMs and Prompting for RL. LLMs have recently demonstrated significant potential in RL. Early works [24]–[28] leverage pretrained foundation models to generate reward signals for RL agents, but these approaches often require frequent queries to LLMs, resulting in high token consumption and reduced training efficiency due to the large number of environment samples needed. More recent studies [8], [9], [16], [29], [30] focus on enabling robots to acquire low-level skills by designing effective reward functions, while others [31], [32] introduce CoT prompting to enhance LLM reasoning for improved reward design. However, these methods still face challenges in adapting to complex, multi-step tasks. Instead, RE-GoT utilizes graph-based thinking, similar to how humans solve complex problems, empowering LLMs with comprehensive analysis and reasoning capabilities to generate more effective rewards. Furthermore, RE-GoT replaces human intervention with VLMs, accelerating the reward search process.

III. PRELIMINARY

Problem Setup. We define our robot agent as a Markov Decision Process (MDP), represented by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma, \rho_0)$, where the environment consists of a state space $s \in \mathcal{S}$ and an action space $a \in \mathcal{A}$. The transition model $\mathcal{P}(s' \mid s, a)$ governs the environment dynamics while ρ_0, $\mathcal{R}$ and γ are the distribution of the initial robot state, the environment's reward function, and the discount factor, respectively. We aim to get an optimal policy $a_t \in \pi_{\mathcal{R}_\theta}(a_t \mid s_t)$ given the parameterized reward function $\mathcal{R}_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ by maximizing the expected reward:

$$\pi_{\mathcal{R}_\theta} = \arg\max_{\pi} \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}_\theta(s_t, a_t) \right] \tag{1}$$

where T denotes the trajectory length, and we define the rollout trajectory over the optimal policy $\pi_{\mathcal{R}_\theta}$ as follows:

$$\xi(\pi_{\mathcal{R}_\theta}) \sim \rho_0(s_0) \prod_{t=0}^{T-1} \mathcal{P}(s_{t+1}|s_t, a_t)\, \pi(a_t|s_t) \tag{2}$$

We consider a framework where LLMs are provided with a textual description $\mathcal{T}$ and a graph structure $\mathcal{G}$ of the task, while VLMs receive the rollout video of a trained RL agent $\mathcal{D}(\xi(\pi_{\mathcal{R}_\theta}))$, $\xi(\pi_{\mathcal{R}_\theta}) \in \Xi$. Given an observation model $\mathcal{D} : \Xi \to \mathcal{D}$ mapping rollout trajectory to the observation space, our objective is to minimize the difference between the expected and actual behaviors:

$$\min_{\mathcal{R}_\theta} \mathop{\mathbb{E}}_{\xi(\pi_{\mathcal{R}_\theta})} \mathcal{L}\left[E_{LLM}(\mathcal{T}, \mathcal{G}), E_{VLM}(\mathcal{D}(\xi(\pi_{\mathcal{R}_\theta})))\right] \tag{3}$$

Classic Gradient-Based Optimization. The optimization in loss function (3) represents a bi-level problem [33], where

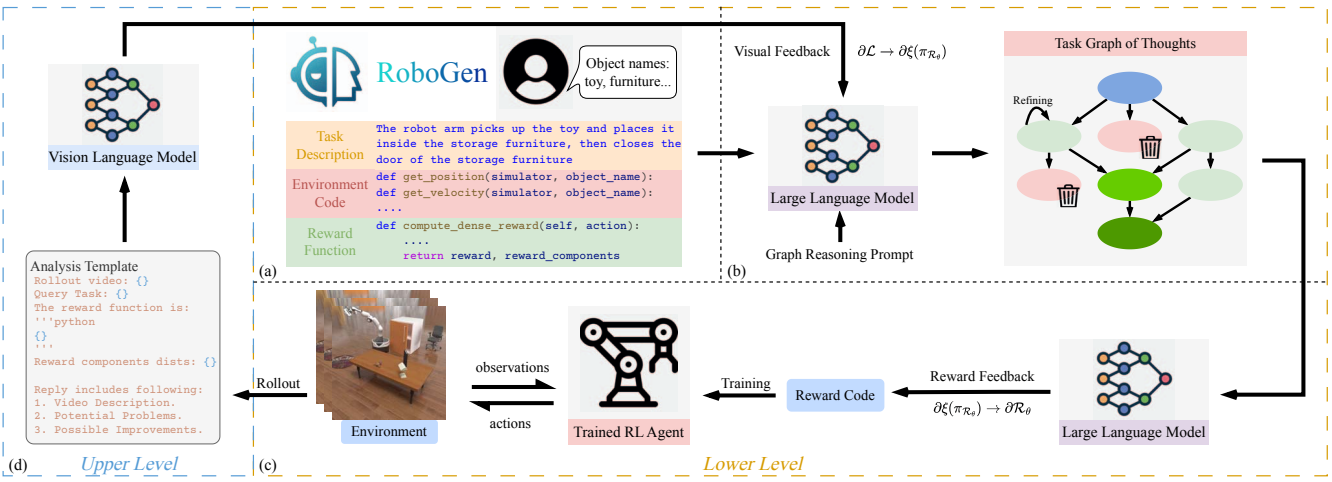

Fig. 2. Overview of the RE-GoT framework. The upper-level evaluates rollout videos using VLMs to provide visual feedback, while the lower-level refines reward functions using LLMs with a graph-based reasoning approach. (a) It prompts LLMs with the environment abstraction to connect to the robotics system. (b) LLMs decompose the task into a text-attributed graph. (c) Given the graph structure and visual feedback, LLMs refine the reward function. (d) VLMs analyze the rollout videos to provide structured feedback on the trained RL agent.

the upper-level optimization minimizes the visual loss of $\xi(\pi_{\mathcal{R}_\theta})$ under policy $\pi_{\mathcal{R}_\theta}$, and the lower-level solves the RL problem to obtain $\pi_{\mathcal{R}_\theta}$. Based on classical gradient-based method, we optimize the object by computing the reward gradient:

$$\nabla_{\mathcal{R}_\theta}\mathcal{L} = \frac{\partial\mathcal{L}}{\partial\xi(\pi_{\mathcal{R}_\theta})} \cdot \frac{\partial\xi(\pi_{\mathcal{R}_\theta})}{\partial\mathcal{R}_\theta} \tag{4}$$

However, before computing the gradient, we need to make sure the loss $\mathcal{L}$ and the observation model $\mathcal{D}$ be explicitly defined and differentiable, which is not feasible due to the need for data preprocessing and explicit modeling. To address this, we propose leveraging the capabilities of VLMs and LLMs as a gradient-free alternative to solving (4). Instead of differentiating through RL training, we frame the problem as aligning expert task knowledge, provided by LLMs in the form of text $\mathcal{T}$ and graph structure $\mathcal{G}$, with the visual rollout $\mathcal{D}(\xi(\pi_{\mathcal{R}_\theta}))$ evaluated by VLMs. This avoids the need for explicitly modeling $\mathcal{L}$ and $\mathcal{D}$, allowing reward learning to be guided by high-level expert priors rather than relying on direct gradient propagation through the reinforcement learning process.

IV. METHOD

A. RE-GoT Overview

Based on the reward gradient (4), for learning a reward with a better performance, we search the space in the direction with the two stages via chain rule, where 1) Visual feedback $\partial\mathcal{L} \rightarrow \partial\xi(\pi_{\mathcal{R}_\theta})$, from structured embedding loss to robot behaviors. 2) Reward update $\partial\xi(\pi_{\mathcal{R}_\theta}) \rightarrow \partial\mathcal{R}_\theta$, from robot behaviors to reward functions. Specifically, the first stage is to minimize the behavior loss by visual feedback, while the second stage informs how to update reward in response to the robot behavior improvement. As it illustrates the RE-GoT framework in Fig. 2, following the bi-level optimization, we divide our system into the upper-level named rollout evaluation (Section IV-B), which focuses on evaluating the rollout videos and providing visual feedback, and the lower-level named reward refinement (Section IV-C), which aims to refine the reward function based on the graph structure of the task and the feedback from the upper-level.

B. Upper-Level: Rollout Evaluation.

The upper level leverages VLMs to evaluate the performance of RL agents by analyzing rollout videos given the analysis template shown in Fig. 2 (d). After the RL agent is trained with the current reward function, several rollout videos are generated to capture its behavior. The VLMs act as an automated evaluator, processing these videos to provide structured, textual feedback on task completion, failure modes, and areas for improvement. This feedback is used to identify discrepancies between the agent's behavior and the intended task objectives, guiding subsequent reward refinement. By automating the evaluation process and reducing reliance on human feedback, this stage accelerates the reward evolution loop and ensures that the agent's learning is aligned with high-level task goals.

C. Lower-Level: Reward Refinement.

1) System initialization: Component Fig. 2 (a) enables the connection of LLMs to the robotics simulator via the environment abstraction, which will be executed only once from the beginning. This abstraction encompasses the observation and action spaces, a detailed task description including all relevant objects, initial reward functions, and a set of callable Python APIs. These APIs provide access to essential environment information, such as the robot's position, object poses, collision detection, etc. Following the principles of task decomposition in RoboGen [32], RE-GoT employs the LLM to categorize each substep as either *primitive* or *reward* according to its control complexity and analytical solvability. Primitive substeps are defined as motions with low contact complexity, which can be reliably executed via motion planning with simulator APIs. In contrast, reward substeps are

designated for stages involving high-dimensional physical interactions where analytical modeling is infeasible and reinforcement learning is required to acquire adaptive policies.

2) Text-attributed graph construction: To better leverage the reasoning capabilities of LLMs, Fig. 2 (b) first enables LLMs to decompose the task into a structured graph representation, providing an organized and interpretable view of the solution space. However, LLMs may sometimes generate incorrect or misleading outputs. To address this, we employ heuristic rules and in-context learning with a few examples to guide the LLM during the decomposition process. Once provided with the task description, the LLM constructs a text-attributed graph, which is then used for subsequent reward refinement. An example of the text-attributed graph represented as its GoT is shown in Fig. 3.

Formally, we define a text-attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_v, \mathcal{T}_e)$, where $\mathcal{V}$ is the set of nodes representing stages, $\mathcal{E}$ is the set of edges denoting transitions between stages, and $\mathcal{T}_v$, $\mathcal{T}_e$ are the sets of attributes associated with each node and edge, respectively. Although the number of steps to achieve the task is fixed for each task, each sub-goal may yield multiple possible outcomes due to variations in actions and environments. The node attributes $\mathcal{T}_v$ describe each stage using the robot, objects, and environment status, while the edge attributes $\mathcal{T}_e$ characterize the robot behaviors required for stage transitions. Similar to human problem-solving, there are often multiple ways to reach the next sub-goal, with varying efficiency and likelihood of success. Unlike prior work [14], our approach does not require manual design of the strategy graph or repeated LLM queries for each node. Instead, we leverage the LLM once to generate the entire graph of thoughts for the task, resulting in a more efficient and resource-saving process.

3) Reward function refinement: Rather than directly generating the reward function, Fig. 2 (c) uses the interfaces from the robotics system as the prior knowledge mentioned in component (a). With all the information fed into the LLM, the reward function refinement process becomes more targeted, allowing the LLM to better understand the system's capabilities and constraints. This enables the LLM to generate reward functions that are both practical and aligned with the specific task objectives, ensuring the robot's behavior is effectively guided by the reward signals.

As we get the visual feedback from VLMs and the text-attributed graph of thoughts from LLMs, we prompt them both with current used reward function to LLMs to refine the reward function in two aspects: 1) Add more constraints, remove redundant constraints or modify current function form to make it more suitable for the current task. 2) For each component within the reward function, search the best weights θ for each component to make the reward function $\mathcal{R}_\theta$ more effective.

```
"text_attributed_graph": {
  "node_idx": [0, 1, 2, 3, 4],
  "node_attr": [
    "idx=0 [S0] Robot is in initial pose.
Button is unpressed.",
    "idx=1 [S1] ..."
    ...
  ],
  "edge_index": [
    [0, 1, 1, 2],
    [1, 2, 3, 4]
  ],
  "edge_attr": [
    "(0->1) grasp the start button",
    "(1->2) ...",
    ...
  ]
}
```

Fig. 3. Example of the text-attributed graph for Press the Start Button, where S_i indicates the index of the sub-goal.

D. Implementation Details

In our framework, we utilize gpt-4o-2024-08-06 as our core reasoning engine, prompted as an expert in robotics and reinforcement learning. The LLM performs a single-query task decomposition into a text-attributed graph consisting of vertices and edges. For each reward substep, the LLM generates reward code, selects the optimal action space, and defines verifiable success conditions based on simulator APIs. For visual supervision, gemini-1.5-pro serves as the VLM evaluator, receiving the video alongside the current reward function and a statistical distribution (maximum, mean, minimum, and standard deviation) of individual reward components. The VLM then provides structured semantic feedback, including a video description, identification of motion problems, and specific reward redesign suggestions.

For policy learning, we adopt Soft Actor-Critic (SAC) as the primary reinforcement learning algorithm for most tasks, while Proximal Policy Optimization (PPO) is employed specifically for the *PickCube* task in ManiSkill2 to ensure stable convergence. Following the protocol in RoboGen, we conduct training for 1×10^6 environment steps per reward substep to ensure policy proficiency. The observation space is composed of low-level environment states, including 6D object poses and joint angles of articulated parts, while the action space is defined as 6D end-effector control. All experiments were conducted on one NVIDIA RTX 4070 Ti Super GPU.

V. EXPERIMENTS

A. Evaluation Setup

1) Objectives: To evaluate our proposed framework, we investigate the following hypotheses through a series of experiments:

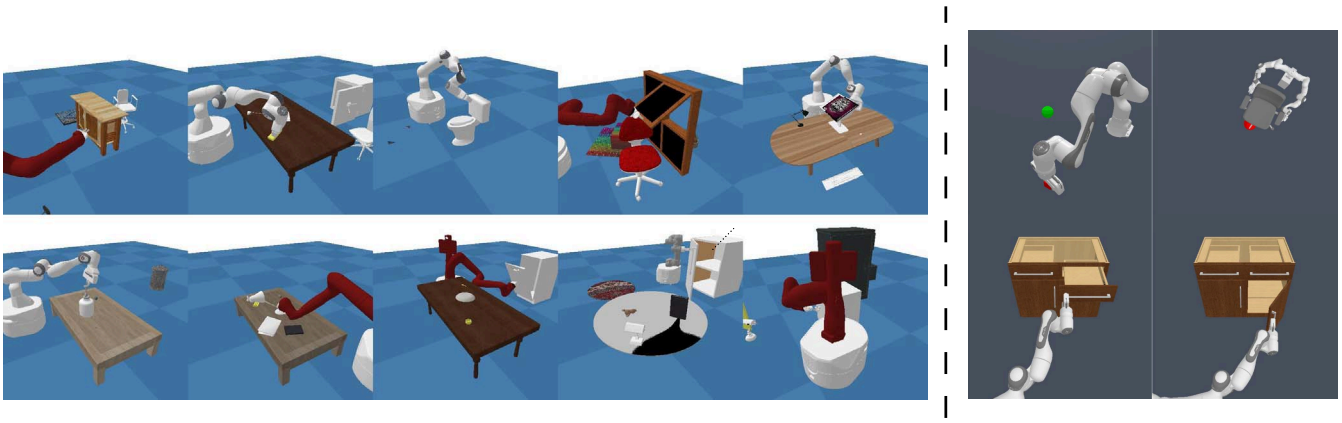

Fig. 4. Evaluation environments. Ten tasks from RoboGen on the left: *Open both table doors*, *Retrieve item from safe*, *Flush toilet*, *Close window*, *Tilt display screen*, *Close dispenser lid*, *Turn on Lamp*, *Load dish into dishwasher*, *Store item into storage*, and *Rotate safe knob*. Four tasks from ManiSkill2 on the right: *PickCube*, *OpenCabinetDrawer*, *OpenCabinetDoor*, and *PushChair*.

$\mathcal{H}_1$ - Can this framework enhance reward functions to achieve effective agents across general manipulation tasks?

$\mathcal{H}_2$ - How does this pipeline perform compared to other LLM-based unsupervised methods?

$\mathcal{H}_3$ - How much impact does RE-GoT have on LLM reasoning compared to unstructured direct prompting, and does in-context learning really help improve the GoT ability?

$\mathcal{H}_4$ - Does in-context learning really help improve the GoT ability to achieve a more stable and precise reward function?

$\mathcal{H}_5$ - Can iterative updates on the graph of the task reduce the numerical imprecision and instability, thereby improving the efficacy?

2) Environment: We evaluate our framework on a diverse suite of 14 robotic manipulation tasks designed to simulate household interaction scenarios. These tasks are sourced from two prominent benchmarks: RoboGen [32] and Maniskill2 [34]. From RoboGen, we adopt 10 tasks spanning a wide range of object interactions and actuator types. These tasks are implemented in PyBullet with extensive randomization across robots and objects properties. We additionally include 4 tasks from ManiSkill2 to further validate the generalization of our framework.

3) Metrics: To evaluate skill learning performance, we use the success rate of the learned policy as the primary metric. This metric quantifies the proportion of successful attempts in achieving the desired task outcome, providing a clear indication of the effectiveness of the learned policy. We also report the mean episode length for each task to provide a more comprehensive understanding of the performance across different tasks.

B. Baseline

We compare our method with three baseline approaches of reward generation with LLM: a) **RoboGen** [32] integrates task proposal, scene generation, training supervision generation, and skill learning into one pipeline, obtaining a trained agent given task scheme or even only objects. It also provides numerous example tasks and allows users to generate custom ones. b) **RewardSelfAlign** [31] (SA) iteratively refines LLM-generated reward functions through self-alignment, updating reward parameters by minimizing ranking discrepancies between LLM-preferred trajectories and the learned reward function. c) **Text2Reward** [9] (T2R) generates dense reward function code using zero-shot or few-shot method by providing the environment abstraction and task description to LLM and refines reward functions given the human feedback to LLM.

We compare the performance in various tasks in RoboGen to examine if RE-GoT could be generalized across different tasks for $\mathcal{H}_1$. In order to answer $\mathcal{H}_2$, we use the reward functions generated by T2R and SA, and reward functions generated by our system to train a RL agent respectively, comparing the training process.

C. Results and Analysis

For evaluation, we report success rates across 10 tasks from RoboGen over 4 random seeds with 10 trials per seed (40 trials per task in total), comparing the performance of the evolved reward functions generated by RE-GoT against the original baselines provided by RoboGen. This comparison highlights the impact of our framework in enhancing reward design for general robotic manipulation tasks, which is

TABLE I
SR (%) COMPARISON ON ROBOGEN TASKS

Substeps	Task	RoboGen	RE-GoT
2	Close Dispenser Lid	25.0±2.9	**90.0±4.1**
	Turn On Lamp	25.0±6.5	**80.0±0.0**
	Rotate Safe Knob	30.0±5.8	**82.5±2.5**
3	Flush Toilet	90.0±0.0	**100.0±0.0**
	Tilt Display Screen	30.0±0.0	**65.0±6.5**
4	Close Window	80.0±0.0	**100.0±0.0**
	Open Both Table Doors	80.0±0.0	**100.0±0.0**
6	Load Dish into Dishwasher	**10.0±5.8**	**10.0±5.8**
	Store Item in Storage	10.0±4.1	**47.5±2.5**
8	Retrieve Item from Safe	42.5±2.5	**70.0±0.0**

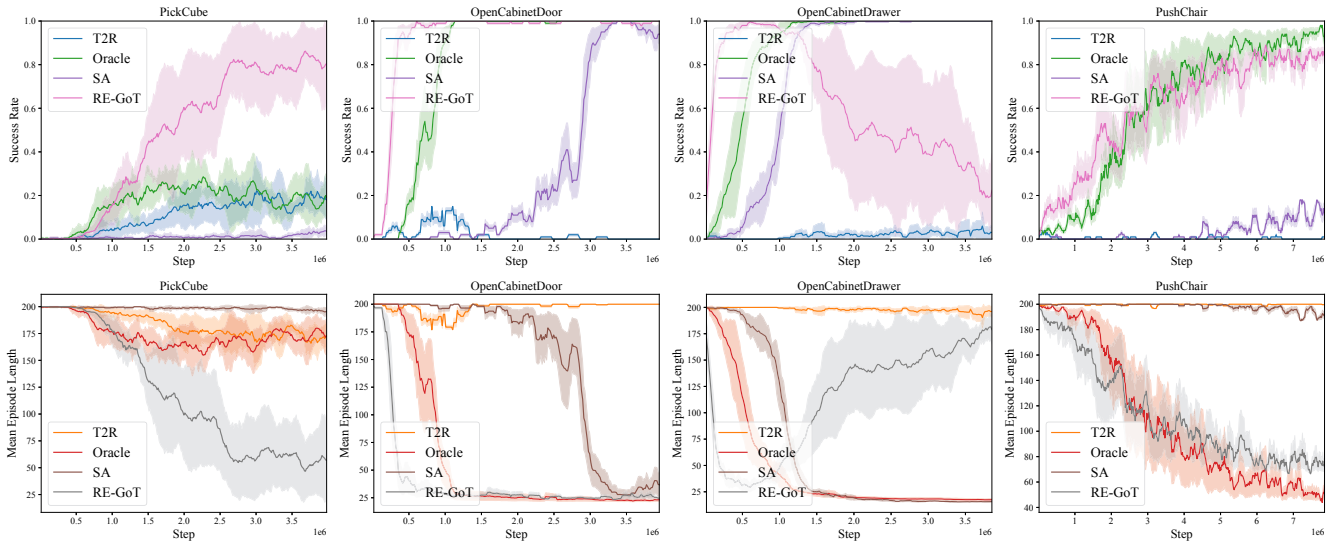

Fig. 5. Success Rate & Average Episode Length vs Exploration Steps on four ManiSkill2 tasks. The solid lines represent the mean, while the shaded areas indicate the standard error of the mean. Oracle means the expert-written reward function provided by the environment.

shown in Table I. We also evaluate four reward functions generated by T2R in zero-shot settings, as well as the best hyperparameter combinations produced by SA. Their performances are compared with ours in terms of success rate and average episode length, as shown in Fig. 5.

1) The enhanced reward achieves an agent with better performance: As shown in Table I, we classify the 10 tasks into 5 categories based on the number of substeps required to complete the task. The success rates of the reward functions generated directly by LLMs are generally lower than those from ours, i.e. an improvement of 32.25% on average which supports $\mathcal{H}_1$. For the task Load Dish into Dishwasher, the randomization for the positions and orientations of the desk and the dishwasher always makes it impossible to open the dishwasher door completely by the robot, which explains no improvement on this task. We observe that while our method achieves the largest improvement on tasks with 2 substeps, its effectiveness remains consistent as the number of substeps increases from 3 to 8. This indicates that our method scales well with task complexity and maintains its advantage over the original reward functions, even in more demanding settings.

2) RE-GoT performs better across different platforms against other LLM-based baselines: To further evaluate the stability and generalization of RE-GoT and address $\mathcal{H}_2$, we conduct experiments on four environments from ManiSkill2, each with five different random seeds, as shown in Fig. 5. We observe that RE-GoT consistently outperforms the other two baselines across all tasks, achieving the best success rates of 86.20%, 100.00%, 99.67%, and 89.00%, respectively. While our method achieves performance comparable to the oracle, it notably improves the success rate for PickCube from 28.60% to 86.20%. However, we also find that the performance of our method on *OpenCabinetDrawer* declines as training continues after step 1×10^6, we suspect that this may be because the VLM fails to analyze the rollout videos correctly and provide accurate feedback.

D. Ablation Study

1) Effectiveness of the GoT Architecture: To answer $\mathcal{H}_3$ and isolate the impact of our framework from the underlying model strength, we compare RE-GoT with a baseline without GoT. In this setting, the LLM is provided with identical environment abstractions and task descriptions but is restricted to unstructured direct prompting. As illustrated in Fig. 6, the performance on *PickCube* and *OpenCabinetDoor* reveals that there is nearly no improvement in success rates without the GoT structure. While these two representative cases are presented for visual clarity, we observed a consistent performance trend across the entire task suite. This result confirms that while state-of-the-art models possess significant internal knowledge, the implementation of a structured architectural scaffolding is a prerequisite for effectively organizing that knowledge to solve manipulation tasks.

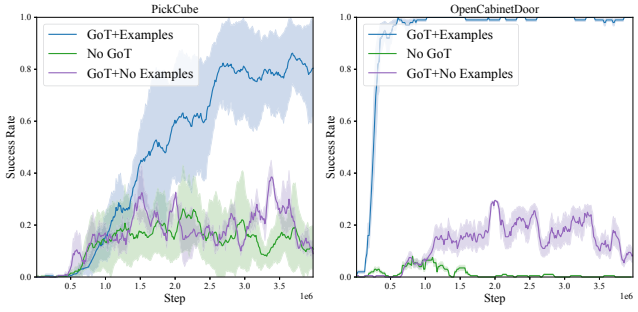

Fig. 6. Effect of different prompting strategies on performance. LLMs achieve the best reward design with GoT and in-context examples.

2) Effectiveness of in-context learning: To evaluate the effectiveness of in-context learning, we compare the perfor-

mance of LLMs in zero-shot and few-shot settings. In the few-shot setting, LLMs are provided with a few examples of task descriptions and graph structures as context, while in the zero-shot setting, no such examples are given. This comparison allows us to assess whether few-shot in-context learning improves the GoT ability, addressing $\mathcal{H}_4$. From Fig. 6, we can conclude that few-shot enables us to greatly improve the reasoning ability of LLMs to generate better rewards.

3) Impact of continuous updating loop: To determine whether periodic updates on the graph help create more stable and precise reward functions, we demonstrate the change in success rate as the number of iterations increases in the pipeline loop, which tries to figure out $\mathcal{H}_5$. As reported in Table II, we observe that the success rate of the tasks generally increases as the number of iterations increases and achieves its peak at 8 iterations. This indicates that the continuous updating loop helps to refine the reward functions and improve the performance of the RL agent.

TABLE II
SR (%) OF TASKS AS RE-GOT ITERATIONS INCREASE

Task \ Iteration	n=2	n=4	n=6	n=8
Turn On Lamp	57.5±2.5	70.0±4.0	75.0±2.5	80±0.0
Flush Toilet	90.0±0.0	95.0±2.9	100.0±0.0	100.0±0.0
Close Window	90.0±0.0	92.5±2.5	100.0±0.0	100.0±0.0

E. Efficiency Discussion

Our method shifts the burden from non-scalable human reward engineering to scalable computational search in simulation. By decomposing long-horizon tasks into structured substeps via the GoT architecture, we significantly narrow the exploration space, enabling policy convergence within 1×10^6 steps per substep. Moreover, to manage costs and latency, we adopt a keyframe sampling strategy on the input video rollouts that minimizes token usage and ensures VLM feedback returns in under 10 seconds. Crucially, this reasoning process occurs strictly offline. Once the reward functions are evolved, the final policy can be deployed online without any LLM querying.

F. Path to Physical Deployment Discussion

Although current evaluations are conducted in simulation, the modular architecture of RE-GoT provides a viable path to physical deployment by bridging the sim-to-real gap. Following the pipeline established in Real2Sim-Eval [35], a high-fidelity digital twin can be reconstructed from real-world environments to ensure visual and physical consistency. Within this digital twin, SAM3D [36] can be utilized for object-level scene reconstruction and segmentation, effectively replacing privileged simulator states with vision-based perception. Crucially, as the VLM-based evaluator operates directly on raw video streams rather than internal simulator code, it remains inherently domain-agnostic and capable of providing feedback in reconstructed real environments. This enables RE-GoT to function as an offline reward evolver within a digital twin, generating robust rewards that facilitate zero-shot or few-shot transfer to physical deployment.

VI. CONCLUSIONS & LIMITATIONS

In this work, we introduced RE-GoT, a novel bi-level framework that leverages GoT and LLMs for reward function evolution in reinforcement learning. By integrating structured graph-based reasoning and visual feedback from VLMs, our approach enables more effective task decomposition, adaptive reward refinement, and reduces reliance on human supervision. Extensive experiments on RoboGen and ManiSkill2 benchmarks demonstrate that RE-GoT consistently outperforms existing LLM-based baselines, achieving higher success rates and better generalization across diverse robotic manipulation tasks. Ablation studies further validate the importance of GoT prompting, in-context learning, and continuous reward refinement. Our results highlight the potential of combining LLMs and VLMs with graph-based reasoning to advance autonomous reward design and improve the scalability and adaptability of RL systems in complex environments.

RE-GoT is not without limitations. The performance of the framework is highly dependent on the quality of the LLMs and VLMs used, as well as the accuracy of the task description and graph structure provided. In cases where the LLMs generate incorrect or misleading outputs, the effectiveness of the reward functions may be compromised. Additionally, while our approach reduces reliance on human supervision, it still requires some level of expert knowledge to define the task description and graph structure. Future work could explore the integration of more advanced LLMs and VLMs for deployment on real robots, as well as the development of more robust prompting strategies to further enhance the performance and generalization of RE-GoT across a wider range of tasks and environments. Furthermore, we plan to create an extensive GoT dataset of different tasks and fine-tune the LLMs to improve the performance of our framework.

REFERENCES

[1] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[2] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[3] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*. PMLR, 2018, pp. 651–673.

[4] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[5] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[6] K. Lee, L. Smith, and P. Abbeel, "Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training," *arXiv preprint arXiv:2106.05091*, 2021.

[7] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[8] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[9] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2reward: Reward shaping with language models for reinforcement learning," *arXiv preprint arXiv:2309.11489*, 2023.

[10] S. Banerjee, A. Agarwal, and S. Singla, "Llms will always hallucinate, and we need to live with this," in *Intelligent Systems Conference*. Springer, 2025, pp. 624–648.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[12] Z. Li, H. Liu, D. Zhou, and T. Ma, "Chain of thought empowers transformers to solve inherently serial problems," *arXiv preprint arXiv:2402.12875*, vol. 1, 2024.

[13] C. Mitra, B. Huang, T. Darrell, and R. Herzig, "Compositional chain-of-thought prompting for large multimodal models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 420–14 431.

[14] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 682–17 690.

[15] Y. Yao, Z. Li, and H. Zhao, "Got: Effective graph-of-thought reasoning in language models," in *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024, pp. 2901–2921.

[16] H. Li, X. Yang, Z. Wang, X. Zhu, J. Zhou, Y. Qiao, X. Wang, H. Li, L. Lu, and J. Dai, "Auto mc-reward: Automated dense reward design with large language models for minecraft," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 426–16 435.

[17] A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine, "Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 281–15 295, 2022.

[18] A. D. Laud, *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[19] S. Singh, R. L. Lewis, and A. G. Barto, "Where do rewards come from," in *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society, 2009, pp. 2601–2606.

[20] A. Y. Ng, S. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, no. 2, 2000, p. 2.

[21] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[22] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv preprint arXiv:1710.11248*, 2017.

[23] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in atari," *Advances in neural information processing systems*, vol. 31, 2018.

[24] Y. Du, K. Konyushkova, M. Denil, A. Raju, J. Landon, F. Hill, N. de Freitas, and S. Cabi, "Vision-language models as success detectors," *arXiv preprint arXiv:2303.07280*, 2023.

[25] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas, "Guiding pretraining in reinforcement learning with large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 8657–8677.

[26] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar, "Minedojo: Building open-ended embodied agents with internet-scale knowledge," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 343–18 362, 2022.

[27] S. Karamcheti, S. Nair, A. S. Chen, T. Kollar, C. Finn, D. Sadigh, and P. Liang, "Language-driven representation learning for robotics," *arXiv preprint arXiv:2302.12766*, 2023.

[28] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," *arXiv preprint arXiv:2303.00001*, 2023.

[29] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[30] S. Sun, R. Liu, J. Lyu, J.-W. Yang, L. Zhang, and X. Li, "A large language model-driven reward design framework via dynamic feedback for reinforcement learning," *arXiv preprint arXiv:2410.14660*, 2024.

[31] Y. Zeng, Y. Mu, and L. Shao, "Learning reward for robot skills using large language models via self-alignment," *arXiv preprint arXiv:2405.07162*, 2024.

[32] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan, "Robogen: Towards unleashing infinite data for automated robot learning via generative simulation," *arXiv preprint arXiv:2311.01455*, 2023.

[33] H. Mahesheka, Z. Xie, Z. Wang, and W. Jin, "Language-model-assisted bi-level programming for reward learning from internet videos," *arXiv preprint arXiv:2410.09286*, 2024.

[34] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao *et al.*, "Maniskill2: A unified benchmark for generalizable manipulation skills," *arXiv preprint arXiv:2302.04659*, 2023.

[35] K. Zhang, S. Sha, H. Jiang, M. Loper, H. Song, G. Cai, Z. Xu, X. Hu, C. Zheng, and Y. Li, "Real-to-sim robot policy evaluation with gaussian splatting simulation of soft-body interactions," *arXiv preprint arXiv:2511.04665*, 2025.

[36] X. Chen, F.-J. Chu, P. Gleize, K. J. Liang, A. Sax, H. Tang, W. Wang, M. Guo, T. Hardin, X. Li *et al.*, "Sam 3d: 3dfy anything in images," *arXiv preprint arXiv:2511.16624*, 2025.