Reward Design Framework Based on Reward Components and Large Language Models

Kexin Jin
School of Control Science and Engineering
Shandong University
Jinan, Shandong, China
kx.jin@mail.sdu.edu.cn

Guohui Tian*
School of Control Science and Engineering
Shandong University
Jinan, Shandong, China
g.h.tian@sdu.edu.cn

Bin Huang
School of Control Science and Engineering
Shandong University
Jinan, Shandong, China
huangbin@sdu.edu.cn

Yongcheng Cui
School of Control Science and Engineering
Shandong University
Jinan, Shandong, China
cuiyc@mail.sdu.edu.cn

Xiaoyu Zheng
School of Control Science and Engineering
Shandong University
Jinan, Shandong, China
202334997@mail.sdu.edu.cn

Abstract*—The application of Large Language Models (LLMs) in the field of robotics has gained widespread attention. Due to their powerful code generation and contextual understanding capabilities, these models can generate reward functions from task commands, prompts, and environmental code, thus aiding robots in acquiring skills through reinforcement learning (RL). However, reward functions generated by existing models often suffer from issues of low executability and success rates, particularly when handling vague or complex task commands. These models usually require multiple iterations of optimization to achieve the desired outcomes. To address this challenge, we propose a reward function generation method based on reward components, which leverages human-level prior knowledge to improve generation efficiency and accuracy. Specifically, we have constructed a task→reward components dataset and fine-tuned a Reward Component Generator (RCG) using this dataset. The RCG then guides the automatic generation of reward functions for reinforcement learning tasks. Experimental results demonstrate that our method significantly improves reward executability, accuracy, and task success rate compared to state-of-the-art approaches. For more information, please visit our project website at: *https://jkx-yy.github.io/RCG/

Keywords-robot manipulation skills; reinforcement learning; LLMs; reward function; reward components

I. Introduction

Large Language Models (LLMs) trained on extensive IoT data possess commonsense physical knowledge, strong code generation capabilities, and natural language understanding abilities. These capabilities enable them to guide robots in learning manipulation skills within reinforcement learning by generating reward functions from task commands and prompts. However, due to limitations in LLMs' command comprehension, the generated rewards often exhibit randomness and inefficiency, requiring multiple adjustments to effectively guide robots in accomplishing the intended tasks. Various LLM-based methods have been explored to enhance the executability and accuracy of reward functions. For example, reference [1] iteratively improves reward generation with human feedback on observed results. While effective, this method is not user-friendly for non-expert users and reduces the autonomy of robot learning. Reference [2] implements mechanisms for autonomous generation and optimization of reward functions, which still face significant challenges when dealing with complex and ambiguous commands. Reference [3] utilizes predefined templates and a small number of examples to improve the success rate and accuracy of reward function generation, however, this approach limits creativity. In summary, this study aims to contribute to two key challenges in reward function generation: (1) Improving robots' ability to interpret complex, domain-specific, and ambiguous commands; (2) Enhancing the autonomy and efficiency of reward function generation.

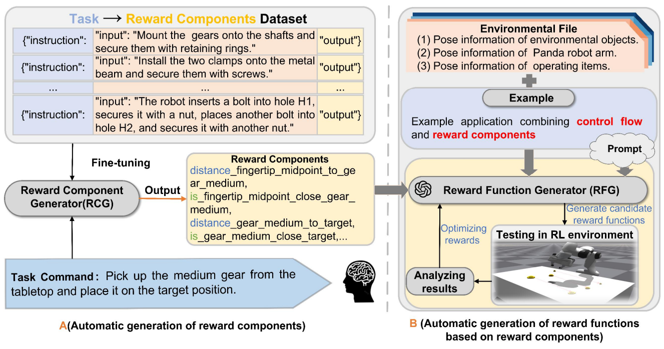

Task ⟶ Reward Components Dataset
{"instruction": "input": "Mount the gears onto the shafts and secure them with retaining rings." "output"}
{"instruction": "input": "Install the two clamps onto the metal beam and secure them with screws." "output"}
... ...
{"instruction": "input": "The robot inserts a bolt into hole H1, secures it with a nut, places another bolt into hole H2, and secures it with another nut." "output"}
Fine-tuning
Reward Component Generator(RCG)
Output
Reward Components
distance_fingertip_midpoint_to_gear_medium,
is_fingertip_midpoint_close_gear_medium,
distance_gear_medium_to_target,
is_gear_medium_close_target,...
Task Command: Pick up the medium gear from the tabletop and place it on the target position.
A(Automatic generation of reward components)
Environmental File
(1) Pose information of environmental objects.
(2) Pose information of Panda robot arm.
(3) Pose information of operating items.
Example
Example application combining control flow and reward components
Prompt
Reward Function Generator (RFG)
Generate candidate reward functions
Optimizing rewards
Testing in RL environment
Analyzing results
B (Automatic generation of reward functions based on reward components)

Fig. 1. Overview of System. The system has two parts: A) RCG generates computation and conversion components from task commands. B) RFG uses these components, environment data, and prompts to create reward functions and perform self-optimization.

This paper proposes a novel reward function generation framework based on reward components (as shown in Fig. 1). This framework centers on reward components and utilizes a reward component generator to parse ambiguous, specialized, and complex task commands into two types of abstract representations: computation components and conversion components. These components guide LLMs in generating reward functions for manipulation tasks. Additionally, control flow statements are introduced to optimize the generation process further. This method significantly improves the efficiency and effectiveness of reward function generation while

1 . National Natural Science Foundation of China under Grant(U22A2059, 62273203)
2 . National Key R&D Program of China under Grant 2018YFB1307101
3 . Taishan Scholars Program of Shandong Province under Grant ts201511005

enhancing the capability of autonomous skill learning in robots, thereby increasing their adaptability to diverse environments and reliability in task execution.

We summarize the three main contributions of this work:

1) This paper constructs a dataset containing 10,000 industrial assembly task commands and their corresponding reward components, covering clear commands, specialized operations, multi-step complex and ambiguous commands. This dataset can be used for fine-tuning LLMs, and supporting research on reward function generation and task planning.

2) A reward function generation framework based on reward components is designed, which possesses enhanced task comprehension capabilities. It can automatically generate task-related reward components and, combined with control flow statements, generate efficient reward functions.

3) Compared to the best research, our method demonstrates a higher success rate in generating instructive reward functions. It can generate rewards with a higher task execution success rate in a shorter time, significantly improving efficiency and effectively guiding robots to complete operational tasks.

II. RELATED WORK

A. Reinforcement Learning for Robotic Manipulation

Reinforcement learning in robotic manipulation enables robots to achieve goals by interacting with their environment. The process involves the robot making decisions based on current state observations, affecting the environment, generating new observations, and using reward feedback to evaluate the effectiveness of actions. The robot then adjusts its strategy based on this feedback and repeats this process. This can be represented by a Markov Decision Process (MDP) $< \mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{r}, \gamma >$, where S is the set of environment and robot states, A is the set of possible actions, $\mathrm{r(s, a)}$ is the reward function, representing the reward the robot receives when taking action a in state s. The reward function provides timely feedback and indirectly guides the robot to learn beneficial actions. The discount factor γ is employed to balance rewards between the present and the future. The goal of reinforcement learning is to learn an optimal policy $\pi^*(a|s)$ that maximizes the cumulative reward. A policy function is typically a neural network that outputs the probability of each action based on the observations. Reinforcement learning is widely used in developing robotic manipulation skills. For instance, reference [4] employs deep reinforcement learning to develop end-to-end policies that map visual inputs to motion outputs, improving the efficiency and accuracy of robotic manipulation tasks. Reference [5] demonstrates how combining human demonstrations with reinforcement learning can address sparse reward problems, successfully validating the insertion task in simulation and implementing it in practice. Reference [6] explores optimizing the synergy of pushing and grasping strategies through reinforcement learning to enhance robotic operation efficiency in cluttered environments. This study aims to teach robotic manipulation skills using deep reinforcement learning.

B. Reward Function Design: Traditional Methods and LLMs

A well-designed reward function helps robots learn desired behaviours, while poorly designed ones can lead to suboptimal or incorrect strategies [7]. Traditional methods rely on expert knowledge and manually defined rules. For instance, sparse rewards have been used to guide robots in grasping tasks by optimizing visual-motor strategies [5]. However, creating sparse rewards for real-world tasks is challenging and often requires reward shaping to provide gradual learning signals, which is a complex but essential process [8]. Methods using LLMs like GPT-4 improve the efficiency of programmatic agents [9]. For instance, reference [2] utilizes encoded LLMs to achieve reward trial-and-error correction and human-level reward generation without manual intervention. Reference [10] introduces modules such as a reward designer, a reward critic, and a trajectory analyzer, which are responsible for creating reward functions, verifying correctness, and analyzing failure causes, respectively. However, these approaches often overlook the issue of insufficient task command comprehension by LLMs and heavily rely on prolonged trial-and-error correction. Some methods attempt to optimize the learning process through real-time human guidance, but they still face significant uncertainties and dependencies [11]. To this end, this paper proposes a reward component method based on task command abstraction, enabling models to achieve a higher-level understanding of task objectives. This approach streamlines the reward function design process and improves the model's efficiency and performance in handling complex, ambiguous task commands.

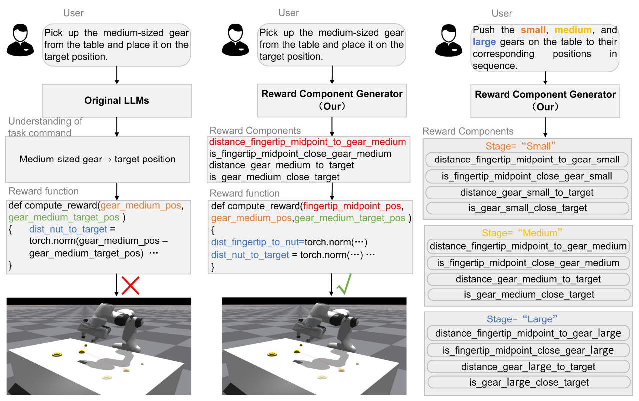
III. METHOD

Fig. 1 illustrates the system framework proposed in this study. The framework consists of two parts: the automatic generation of reward components and the automatic generation of reward functions based on reward components.

A. Automatic Generation of Reward Components

LLMs still have limitations in processing task commands, particularly when faced with vague or unclear dependencies. As shown in Fig. 2, for task commands like "Pick up the medium-sized gear from the table and place it on the target position." LLMs often struggle to automatically infer implicit dependency and sequence relationships (e.g., "manipulators → medium gear" and "medium gear → target position"). They also exhibit instability in identifying missing action subjects. This instability limits the reliability of LLMs in task planning and execution. Furthermore, LLMs often fail to accurately capture the temporal logic in commands with long-term dependencies, such as "Push the large, medium, and small gears on the table to their corresponding positions in sequence." leading to improper sequencing of actions and resulting in disjointed task execution or failure to complete the task as expected.

1) Task → Reward Components Dataset Construction: Humans possess strong understanding and reasoning abilities when processing task commands. To this end, we have constructed a dataset containing 10,000 industrial assembly commands, covering clear, ambiguous, complex, and specialized types of commands. The dataset includes both single-item and multi-item tasks, with over 200 industrial assembly components such as gears, washers, nuts, and nails. Each task command is annotated with computation components and conversion components based on the execution sequence. Computation components are used to calculate the distance between the manipulator and the object or between the object


User
Pick up the medium-sized gear from the table and place it on the target position.
Original LLMs
Understanding of task command
Medium-sized gear→ target position
Reward function
User
Pick up the medium-sized gear from the table and place it on the target position.
Reward Component Generator (Our)
Reward Components
Reward function
User
Push the small, medium, and large gears on the table to their corresponding positions in sequence.
Reward Component Generator (Our)
Reward Components
Stage= "Small"
Stage= "Medium"
Stage= "Large"

```
def compute_reward(gear_medium_pos,
gear_medium_target_pos )
{     dist_nut_to_target =
      torch.norm(gear_medium_pos –
      gear_medium_target_pos) ···
}
```

```
distance_fingertip_midpoint_to_gear_medium
is_fingertip_midpoint_close_gear_medium
distance_gear_medium_to_target
is_gear_medium_close_target
```

```
def compute_reward(fingertip_midpoint_pos,
gear_medium_pos,gear_medium_target_pos )
{
dist_fingertip_to_nut=torch.norm(···)
dist_nut_to_target = torch.norm(···) ···
}
```

```
distance_fingertip_midpoint_to_gear_small
is_fingertip_midpoint_close_gear_small
distance_gear_small_to_target
is_gear_small_close_target

distance_fingertip_midpoint_to_gear_medium
is_fingertip_midpoint_close_gear_medium
distance_gear_medium_to_target
is_gear_medium_close_target

distance_fingertip_midpoint_to_gear_large
is_fingertip_midpoint_close_gear_large
distance_gear_large_to_target
is_gear_large_close_target
```

Fig. 2. Comparison of reward generation: original LLMs vs. our method.

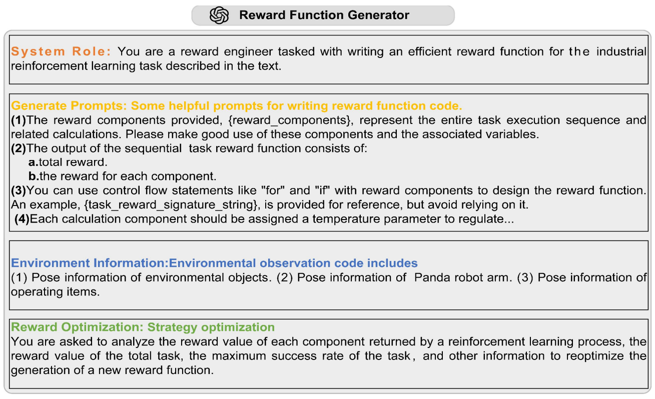
and the target, serving as the basis for the reward function to help achieve the goal. Conversion components are used to determine whether the subtask is complete, supporting the segmented design of the reward function. This dataset accurately covers a variety of task types while also considering the understanding of non-expert inputs. Each record consists of three parts: "Instruction"—listing all distance calculations and necessary condition checks in the robot's task execution sequence; "Input"—refers to task command entered by humans; and "Output"—the standard reward components output for the task, including computation and conversion components.

2) *Reward Component Generator (RCG):* Based on the constructed Task→Reward Components dataset, this paper finetunes the open-source Large Language Model Meta AI (LLaMA) [12] to learn the precise mapping between task commands and reward components, creating an efficient Reward Component Generator (RCG). The design goal of RCG is to simulate human understanding of natural language commands, accurately parse the intrinsic logical relationships within commands, and handle ambiguous commands, thereby reducing human intervention and significantly enhancing the system's autonomy and process automation level.

B. *Automatic Generation of Reward Functions*

Inspired by reference [13], where if-else and loop statements invoke action primitives for embodied control, this paper explores integrating control flow with reward components to optimize task execution in dense reward function design. Then, provide a detailed explanation of the process for automatically generating and optimizing reward functions.

1) *Integration of Control Flow and Reward Components:* For a task command with multiple dependencies, such as "Push the medium and large gears on the table in order to the target position." the task involves two steps: first, pushing the medium gear, and then pushing the large gear. The original rewards generated by LLMs typically take the following form: *total_reward=k1*reward_finger_to_medium_gear+k2*reward_medium_gear_to_target+k3*reward_finger_to_large_gear+k4*reward_large_gear_to_target.* Adjusting these four parameters to ensure the task is executed as expected is a time-consuming process. By integrating conversion components


Reward Function Generator

System Role: You are a reward engineer tasked with writing an efficient reward function for the industrial reinforcement learning task described in the text.

Generate Prompts: Some helpful prompts for writing reward function code.
(1) The reward components provided, {reward_components}, represent the entire task execution sequence and related calculations. Please make good use of these components and the associated variables.
(2) The output of the sequential task reward function consists of:
a.total reward.
b.the reward for each component.
(3) You can use control flow statements like "for" and "if" with reward components to design the reward function. An example, {task_reward_signature_string}, is provided for reference, but avoid relying on it.
(4) Each calculation component should be assigned a temperature parameter to regulate...

Environment Information:Environmental observation code includes
(1) Pose information of environmental objects. (2) Pose information of Panda robot arm. (3) Pose information of operating items.

Reward Optimization: Strategy optimization
You are asked to analyze the reward value of each component returned by a reinforcement learning process, the reward value of the total task, the maximum success rate of the task, and other information to reoptimize the generation of a new reward function.

Fig. 3. Details for Reward Function Generator.

with if-else statements for hierarchical reward design, the task can be ensured to follow the predetermined sequence from the outset, improving the accuracy and efficiency of task execution.

2) *Reward Function Generation (RFG):* As shown in Fig. 3, we have designed four effective prompts for the RFG. By precisely designing the input content, we optimize the AI system's understanding of task requirements, thereby improving the efficiency of reward function generation. First, the system is assigned the role of a "reward design engineer," to design efficient reward functions for industrial assembly tasks. The system understands the task context, current environment state, and its physical-world performance by receiving environmental information. To ensure that the generated reward functions have high executability and success rates, we have designed meaningful prompts. Specifically, the system uses a combination of computation components, conversion components, and control flow statements to guide the execution order of the task, ensuring that each step is performed correctly and providing a simple reference example. Inspired by reference [2], A policy optimization process is introduced to achieve automatic tuning of the reward function. This allows the system to adjust the reward mechanism based on real-time feedback, optimizing task execution efficiency and accuracy.

3) *Reward Function Optimization:* Although using reward components to guide the generation of reward functions can significantly improve the executability, accuracy, and success rate of generated rewards (as described in Section IV-B), these functions may still be suboptimal. To further optimize the process, we introduced a self-feedback mechanism, allowing the generator to dynamically adjust its strategy based on data from each iteration. Specifically, in each iteration, the generator records key data from N reward candidates, such as total rewards, distribution of reward components values, and task success rates. Using the analytical and computational capabilities of RFG to process these data, the generator refines its strategy by optimizing parameters or modifying code to enhance reward performance.

IV. EXPERIMENTS

This section validates the effectiveness of the proposed method through simulations, evaluating the accuracy of reward

TABLE I. TASK COMMAND INFORMATION

Task Name	Task Command	Action	Task Understanding(RCG)	Reward Components(RCG)
Approach_Nut	Move closer to the nut on the table.	Approach	finger→nut	distance_finger_to_nut,is_finger_close_nut
Pick_Nut	Pick up the nut from the table to the target position	Approach Pick	finger→nut nut→target	distance_finger_to_nut,is_finger_close_nut distance_nut_to_target,is_nut_close_target
Nut_Bolt	Pick up the nut from the table and place it on the bolt	Approach Pick,Place	finger→nut nut→bolt	distance_finger_to_nut,is_finger_close_nut distance_nut_to_bolt,is_nut_close_bolt
Nut_Bolt_II	Pick up the nut from the table, raise it 0.1 m, and place it on the bolt.	Approach Pick Lift,Place	finger→nut nut→target nut→bolt	distance_finger_to_nut,is_finger_close_nut distance_nut_to_target,is_nut_close_target distance_nut_to_bolt,is_nut_close_bolt
Push_Gear	Push the medium gear on the table to the target position	Approach Push	finger→medium gear medium gear→target	distance_finger_to_gear,is_finger_close_gear distance_gear_to_target,is_gear_close_target
Push_Gears	Push the medium and large gears on the table in order to reach the target position	Approach Push Approach Push	finger→gear medium gear medium →target1 finger→gear large gear large →target2	distance_finger_to_gear_medium, is_finger_close_gear_medium distance_gear_medium_to_target1, is_gear_medium_close_target1….

components generation under different types of commands and analyzing the advantages of reward function generation based on reward components. The generation of reward components, reward functions, and the optimization of reward functions can be viewed on the project website.

Experimental Environment and Setup: We used the Isaac Gym [14] simulator for environment modeling, a GPU-accelerated physics simulation platform developed by NVIDIA, designed to provide efficient, multi-robot simulation and realistic physical phenomena for reinforcement learning and robotics research. The experiment included two main tasks: *GearsPush* and *NutBolt* (see Fig. 4), each containing multiple operation goals, including both single-step and multi-step tasks (specific tasks are listed in Table I). The Franka Emika Panda robot performed the tasks. All experiments were conducted on a platform with an NVIDIA RTX A5000 GPU, running Ubuntu 20.04, and using the PyTorch framework.

A. Reward Components Generation Result

To thoroughly evaluate the proposed reward components generation method, we present the corresponding results generated from task commands provided by non-expert users in Table I. These task commands are designed to be diverse and representative, covering main operation types (such as approaching, grasping, placing, pushing, and moving) and their combinations. Additionally, the commands encompass typical industrial assembly scenarios. The results in Table I reflect the accuracy of the reward components generation method in handling various task commands, providing data for further optimization of the reward function. This accuracy is crucial for refining the approach. Accurate reward components are highly robust and adaptable to various real-world tasks.

B. Experiment on Reward Functions Generation

We evaluated the tasks in Table I to test the GPT-4-0613-based RFG. Each task underwent five iterations, generating 10 reward function candidates per iteration, with five random seeds for repetition. The A2C [15] algorithm was used to train the robot's policy.

1) Baselines: We employed three baseline methods for comparison. The first was manually designed reward functions, based on variations of the original benchmark task rewards, representing expert-level design. The second was the Eureka [2]

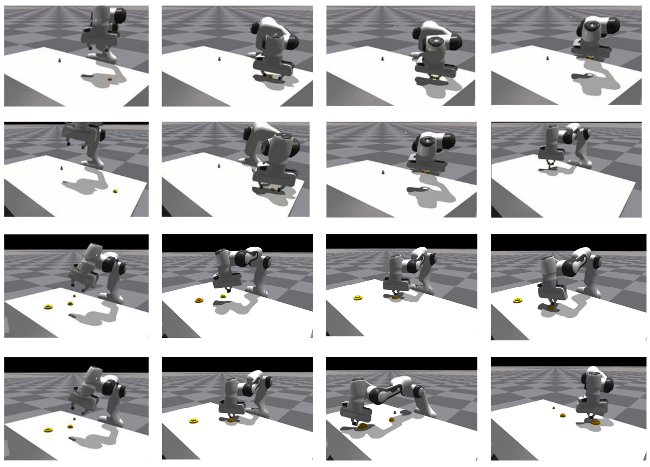


Fig. 4. Visualization of the process of guiding robot operations through component-based rewards.Video keyframes are displayed row by row: Pick_Nut(row 1), Nut_Bolt_II(row 2), Push_Gear(row 3), Push_Gears(row 4).

which generated reward functions from task commands and improved them through self-optimization. The third was a template-based method, similar to L2R [3], for which we developed a dedicated template to ensure fair comparison.

2) Quantitative Metrics: We evaluated the effectiveness of the method using three metrics: **Executability Rate (ER)**, which measured the syntactical correctness and compatibility of the reward function, calculated as the ratio of executable reward functions to the total generated; **Accuracy (AC)**, which assessed the alignment of the reward function with task logic, calculated as the ratio of executable and accurate reward functions to the total generated; **Task Execution Success Rate (TESR)**, which measured the effectiveness of the reward function in guiding the agent to achieve the task goal.

3) Simulation Results: In the experiment, we used reward components generated by RCG to guide RFG in constructing reward functions and validated its effectiveness across multiple tasks. Fig. 4 shows the visualization of four tasks, including key steps and final states.

The experimental results shown in Fig. 5 demonstrate that, across six tasks, the proposed method significantly outperforms traditional automated methods in both ER and AC. Reward

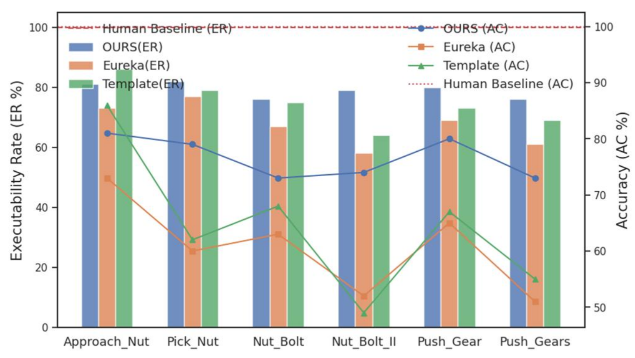


Fig. 5. Average Executability Rate (ER) and Accuracy (AC) across five iterations for each task.

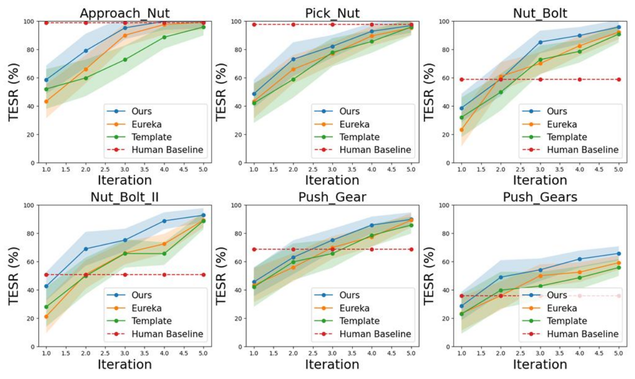


Fig. 6. The impact of the optimal reward function on task execution success rate (TESR) across iterations.

functions designed by experts achieve 100% in these two metrics, indicating excellent syntactical correctness and task logic consistency. In contrast, automated methods often struggle to reach the same level due to potential syntax errors. The proposed method effectively integrates reward components, reducing the use of irrelevant variables, thus improving ER. While the generated reward functions are executable in terms of syntax, their logical accuracy may be insufficient when handling complex or ambiguous commands, leading to task learning failure. However, with the application of human-designed reward components, the method maintains high AC in both simple and complex tasks.

Fig.6 shows that the proposed method significantly outperforms other methods in TESR. After several iterations, the automatically generated reward functions gradually surpass those designed by humans. This is because humans face challenges when adjusting sub-process parameters, as small changes can lead to significant impacts. In contrast, LLMs can quickly optimize parameters by leveraging accumulated data. Among the three automated methods, the proposed method performs the best, not only due to the higher ER and AC of the reward candidates generated in each iteration but also because it excels at adapting to dynamic task changes, continuously optimizing the reward function design, accelerating the iteration process, and significantly improving task completion efficiency and quality.

V. Conclusions

This paper proposes an efficient reward design framework based on reward components. We constructed a dataset of 10,000 task → reward components for common industrial assembly tasks and fine-tuned the reward component generator to handle complex, ambiguous commands and enable precise, logical planning, thereby providing clear guidance for reward generation. Experimental results validate the superiority of this method, offering a more efficient solution for robot skill learning.

References

[1] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong et al., “Text2Reward: Reward Shaping with language models for reinforcement learning.” in The Twelfth International Conference on Learning Representations, 2024.

[2] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman et al., “Eureka: Human-level reward design via coding large language models,” in The Twelfth International Conference on Learning Representations, 2024.

[3] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas et al., “Language to rewards for robotic skill synthesis,” in Proceedings of The 7th Conference on Robot Learning, Proceedings of Machine Learning Research, vol. 229, pp. 374-404, November 2023.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” J. Mach. Learn. Res., vol. 17, pp. 39:1–39:40, 2015.

[5] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot et al., "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” unpublished.

[6] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong, “Efficient learning of goal-oriented push-grasping synergy in clutter,” IEEE Robotics and Automation Letters, vol. 6, pp. 6337–6344, 2021.

[7] S. Booth, W. B. Knox, J. A. Shah, S. Niekum, P. Stone, A. Allievi et al., “The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications,” in AAAI Conference on Artificial Intelligence, 2023.

[8] R. Sutton and A. Barto, “Reinforcement learning: An introduction,”IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 1054–1054,1998.

[9] N. Shinn, B. Labash, and A. Gopinath, “Reflexion: an autonomous agent with dynamic memory and self-reflection,” in Advances in Neural Information Processing Systems, vol. 36, pp. 8634–8652, 2023.

[10] H. Li, X. Yang, Z. Wang, X. Zhu, J. Zhou, Y. Qiao et al., “Auto mc-reward: Automated dense reward design with large language models for minecraft,” 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 16 426–16 435, 2023.

[11] B. Xiao, Q. Lu, B. Ramasubramanian, A. Clark, L. Bushnell, and R. Poovendran, “Fresh: Interactive reward shaping in high-dimensional state spaces using human feedback,” in Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, p. 1512–1520, 2020.

[12] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix et al., “Llama: Open and efficient foundation language models,” unpublished.

[13] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter et al., “Code as policies: Language model programs for embodied control,” in 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 9493–9500, 2023.

[14] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller et al., “Isaac gym: High performance GPU based physics simulation for robot learning,” in Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley et al., “Asynchronous methods for deep reinforcement learning,” in Proceedings of The 33rd International Conference on Machine Learning, vol. 48, pp. 1928–1937, Jun 2016.