

# Boosting Universal LLM Reward Design through Heuristic Reward Observation Space Evolution

Zen Kit Heng<sup>†</sup>, Zimeng Zhao<sup>†</sup>, Tianhao Wu<sup>‡</sup>, Yuanfei Wang<sup>‡</sup>, Mingdong Wu, Yangang Wang, Hao Dong

**Abstract**— Large Language Models (LLMs) are emerging as promising tools for automated reinforcement learning (RL) reward design, owing to their robust capabilities in commonsense reasoning and code generation. By engaging in dialogues with RL agents, LLMs construct a Reward Observation Space (ROS) by selecting relevant environment states and defining their internal operations. However, existing frameworks have not effectively leveraged historical exploration data or manual task descriptions to iteratively evolve this space. In this paper, we propose a novel heuristic framework that enhances LLM-driven reward design by evolving the ROS through a table-based exploration caching mechanism and a text-code reconciliation strategy. Our framework introduces a state execution table, which tracks the historical usage and success rates of environment states, overcoming the Markovian constraint typically found in LLM dialogues and facilitating more effective exploration. Furthermore, we reconcile user-provided task descriptions with expert-defined success criteria using structured prompts, ensuring alignment in reward design objectives. Comprehensive evaluations on benchmark RL tasks demonstrate the effectiveness and stability of the proposed framework. Code and video demos are available at [jingjjjjie.github.io/LLM2Rewards](https://jingjjjjie.github.io/LLM2Rewards)

## I. INTRODUCTION

Simulation is of paramount importance in scaling up robotic learning, as it provides more controllable environments for acquiring diverse robotic skills [24] and offers additional guidance for effective sim-to-real transfer [15]. Traditional methods rely on manual reward shaping, which is labor-intensive and may not generalize well across different tasks and environments [18]. Moreover, poorly designed rewards can hinder the learning process or lead to unintended behaviors, complicating the development of robust robotic systems [13].

Recent advancements in Large Language Models (LLMs) [2] have opened new avenues for automated reward design. LLMs are believed to encapsulate vast amounts of human knowledge and logic from their training corpus. Consequently, the embodied AI community regards LLMs as generalizable encyclopedias, guiding agents to learn skills universally across diverse environment-robot settings under

Zen Kit Heng, Tianhao Wu, Yuanfei Wang, Mingdong Wu and Hao Dong are with the Center on Frontiers of Computing Studies, School of Computer Science, Peking University, Beijing 100871, China, also with PKU-Agibot Lab, School of Computer Science, Peking University, Beijing 100871, China, and also with National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing 100871, China. Zimeng Zhao and Yangang Wang are with School of Automation, Southeast University, Nanjing, China.

<sup>†</sup> The first two authors contributed equally.

<sup>‡</sup> The second two authors contributed equally.

Corresponding to [hao.dong@pku.edu.cn](mailto:hao.dong@pku.edu.cn)

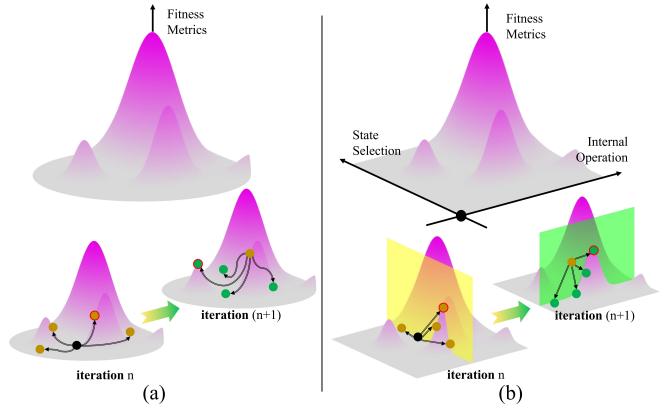


Fig. 1. Comparison diagram of evolutionary process.(a) Eureka’s evaluation and sampling. (b) Our evaluation and sampling.

an imitation learning [10], [6] or reinforcement learning [32], [14], [27], [29] formulation.

Focusing on the latter, this work aims to build a universal RL reward design scheme with LLMs. The primary challenge lies in grounding [3]: making RL agents’ perceptions understandable for LLMs’ planning and ensuring that LLMs’ code is actionable for RL agents’ control. Formally, LLMs design an RL task reward through constructing a corresponding *Reward Observation Space (ROS)*. This space contains a subset of all available environment-robot states, along with the operations defined upon those subset members. Our key idea is to regard the LLM reward design problem as a heuristic sampling to evolve the **ROS** driven by both curiosity and success feedback.

Limited by LLM context length, pioneer works [14], [33] only adopted the local optima **ROS** of the last iteration as a sampling guidance in the current iteration. In pursuit of more thorough and efficient communications, we first incorporate a state execution table accessible to all iterations and samples, breaking the Markovian constraint in LLM dialogues. This table keeps track of historical usage frequency and success contributions of each state in the RL task, ensuring that novel **ROS** configurations are explored by encouraging LLMs to prioritize states with higher success contributions and fewer usages. To further optimize this process, we employ the following strategies: (i) We disentangle the design of space member selection and internal member operation into two sub-problems, alternating between them across iterations, reducing complexity and improving comparability within **ROS** during each iteration. (ii) To prevent LLMs from being misled by local optima, the full reward code is only provided if the success rate exceeds an adaptive threshold.

Otherwise, the reward code is truncated [1]. (iii) Execution errors are mitigated by only updating the state execution table with successful runs, keeping LLMs focused on valid state configurations.

We further get insights on conveying structured and non-controversial information [29], [32], [30] to LLMs. Current formulation [14], [33] takes the design mission from a *user's* description text, while iteratively evaluating each design sample according to another *expert's* success code. As a potential risk, the intentions of these two individuals may be misaligned or even contradictory. To this end, we resort to LLM to reconcile their potential contradictions by prompting with their structured templates. Surprisingly, even for a new task without a definition of success, our framework could facilitate LLM in writing an expert-level success function according to the above templates and user task description. Afterward, the reconciled description becomes the LLM's mission, and the reconciled success replaces the vanilla success. For a fair comparison with the existing approaches [14], [32], [33], our LLM preprocessing is isolated from the subsequent design iterations and the reconciled success is still invisible during LLM design. In our comparison, their performance on reconciled settings has also been evaluated.

In summary, our main contributions are:

- A heuristic **ROS** evolution framework, boosting the performance of LLM reward design.
- A table-based exploration caching mechanism, breaking the Markovian constraint in LLM dialogues.
- A user-expert reconciliation strategy, filling the cognitive gap between users and experts to the same task.

## II. RELATED WORK

**Reward Design Problem.** Rewards are the primary mechanism through which agents learn desirable behaviors in a given environment [8], and their formulation critically affects the efficiency and success of the learning process [12]. Historically, reward functions have been hand-crafted by domain experts for specific tasks like robot navigation or games [17], [23]. However, crafting effective reward functions is challenging, as it requires balancing exploration with enough guidance to avoid suboptimal behavior [25]. For more complex tasks, such as robotic manipulation involving continuous control, reward design becomes significantly more challenging [4], [7]. To address the above challenges, researchers have explored automatic reward generation techniques. Inverse reinforcement learning (IRL) [5], [19] aims to infer reward functions from expert behavior rather than relying on manual design. IRL has demonstrated success across various domains, yet it often demands high-quality expert demonstrations, which can be costly and time-consuming to obtain. Moreover, the inferred reward functions are not always unique or interpretable, leading to ambiguity when applied to new contexts [34]. Evolutionary algorithms represent another approach to reward design. Early works, such as [20], utilized these algorithms to iteratively optimize reward functions by selecting the most promising candidates. Although promising,

these methods are computationally intensive and require predefined templates for possible reward structures, which constrains their flexibility. The recent advent of large language models (LLMs) [21], [28] has expanded the potential for automatic reward generation. For example, Eureka framework [14] generates reward functions from raw environment descriptions and task code. However, the quality of rewards produced by Eureka can be inconsistent, and the generation process may suffer from instability particularly when applied to tasks requiring fine-tuned control. By incorporating advanced evolutionary algorithms, our approach ensures stable reward generation, especially in environments with a high degree of freedom action space [7].

**Scaling up Robotic Learning in Simulations.** Simulations provide a controlled environment where robots can acquire diverse skills with minimal risk and cost, making it a fundamental tool for research in embodied AI [26]. However, despite the advances in simulation platforms, scaling up robotic learning presents several hurdles, primarily in the generation of diverse tasks. Automating task generation is therefore a vital step toward scaling up learning processes in simulation. A key issue in scaling up robotic learning is the creation of diverse and meaningful tasks within simulation environments. Traditionally, task generation has relied heavily on manual design, where experts must define each task, its associated assets, and the reward functions that guide the robot's learning process. This approach is not only labor-intensive but also limits the scope and diversity of tasks that can be generated. One promising direction to scale up robotic learning is through multi-task and meta-reinforcement learning [11], [31], [7]. In multi-task RL, robots learn multiple tasks simultaneously, sharing knowledge across tasks to improve generalization. Frameworks like Meta-World [31] provides a collection of manipulation tasks that encourage robots to develop transferable skills. Meta-reinforcement learning goes further by training robots to learn how to learn, enabling them to quickly adapt to new tasks based on previous experience. Some attempts [9] aims to optimize a robot's policy in such a way that it can rapidly adapt to new, unseen tasks with minimal additional training. These approaches are essential for developing generalist robots that can operate in diverse, dynamic environments. This work aims to create diverse, user-described tasks within existing simulation environments, promoting multi-task learning and expanding the applicability of current simulations.

## III. PROBLEM FORMULATION

We address the reward design problem (RDP) for general RL environments [24], which can be formalized as a tuple  $\text{RDP} \triangleq \langle M, R, \Lambda, \mathcal{F}_{ft} \rangle$ . Given a world model  $M$ , the optimal reward function  $R^*$  is expected to efficiently guide an RL algorithm  $\Lambda$  to accomplish the task:

$$R^* = \arg \max_R \mathcal{F}_{ft}[\Lambda(R)] \quad (1)$$

**World model**  $M$  defines an environment-robot setting including state  $S \in \mathcal{S}$ , action  $A \in \mathcal{A}$  and transition  $\mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ . A task  $\Gamma_M$  defined in  $M$  is to achieve a clear goal. From user's

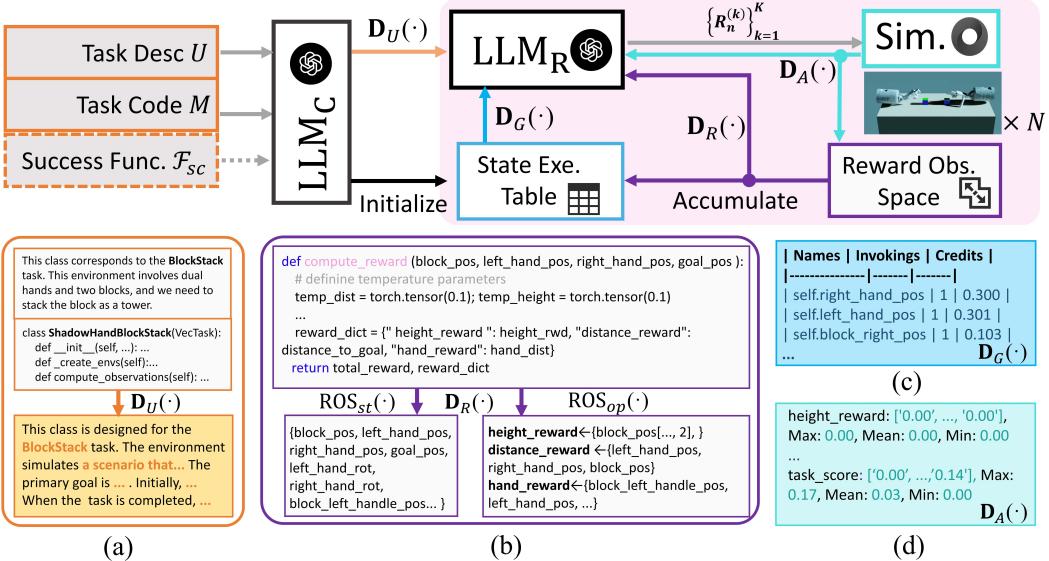


Fig. 2. The pipeline of our proposed framework for heuristic Reward Observation Space (ROS) evolution in LLM-aided RL reward design. (a) User-expert Mission Reconciling. (b) Observation Space Disentanglement. (c) Reward State Execution. (d) Reward Item Performance.

perspective, this goal is expressed through the task description  $U$  (e.g.“make humanoids stand up”). From expert’s perspective, it is grounded as the success function  $\mathcal{F}_{sc} : \mathcal{S} \mapsto \{0, 1\}$  (e.g.“take 1 when the humanoid torso is above a certain height otherwise 0”).

**Reward function**  $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$  compactly quantifies the contribution of every  $\langle S, A \rangle$  pattern to  $\Gamma_M$  with a scalar. When  $R$  is represented by a logic code, its input is a subset of  $\mathcal{S}$ , and its output scalar is obtained by operations within the subset states.

**RL algorithm**  $\Lambda$  explores the optimal policy  $\pi = \pi_\Lambda^*$  guided by a form of  $R$  to complete  $\Gamma_M$ . An interaction episode between  $\pi_\Lambda$  and the  $M$ ’s transition result in a Markov Decision Process. The huge exploring space causes  $\Lambda$  to have no guarantee that  $\pi_\Lambda^*$  will be found on every run, even if the same  $R$  is adopted.

**Fitness function**  $\mathcal{F}_{ft} : \Lambda(R) \mapsto \mathbb{R}$  crucially controls the optimization in Eqn. 1 by measuring how well a form of  $R$  can guide  $\Lambda$  to complete  $\Gamma_M$ . A naive setting is  $\mathcal{F}_{ft} = \mathcal{F}_{sc}$ . However, the optimization guided purely by  $\mathcal{F}_{sc}$  lack effectiveness since  $\mathcal{F}_{sc}$  is usually sparse [14].

In essence, RDP Eqn. 1 requires an expert-crafted  $\mathcal{F}_{ft}$  to guide the optimization of  $R$  effectively.

#### IV. METHODOLOGY

In this paper, we treat the dialogue between LLM and  $\Lambda(R)$  as an iterative evolutionary process similar to Eqn. 1. This allows users without domain expertise to design rewards using natural language  $U$  while extending the guidance beyond  $\mathcal{F}_{ft}$  alone. Specifically, the LLM is prompted to design  $K$  reward samples  $\mathcal{R}_n \triangleq \{R_n^{(k)}\}_{k=1}^K$  at the  $n$ -th iteration ( $n = 1, \dots, N$ ):

$$\begin{aligned} \mathcal{R}_n &= \text{LLM}_R (\mathbf{D}_R(R_{n-1}^*), \mathbf{D}_A[\Lambda(R_{n-1}^*)], \\ &\quad \mathbf{D}_G(\sum_m^{n-1} \mathcal{R}_m) \mid \mathbf{D}_U(U)) \\ R_n^* &= \arg \max_{R \in \mathcal{R}_n} \mathcal{F}_{sc}[\Lambda(R)] \end{aligned} \quad (2)$$

Initially,  $\mathbf{D}_U(\cdot)$  interprets the task description  $U$  (IV-A). In subsequent iterations (IV-B), composite guidance is applied.  $\mathbf{D}_R(\cdot)$  maps  $R$  into its *Reward Observation Space (ROS)*.  $\mathbf{D}_A(\cdot)$  summarizes the performance of  $\Lambda$ .  $\mathbf{D}_G(\cdot)$  acts as memory, transcribing all historical reward samples into a *State Execution Table*. The target  $R^*$  is approximated by selecting the best-performing local optimum samples over  $N$  iterations:

$$R^* \approx \arg \max_{R \in \{R_1^*, \dots, R_N^*\}} \mathcal{F}_{sc}[\Lambda(R)] \quad (3)$$

##### A. User-expert Mission Reconciling

Two types of human involvement influence 2: users describe tasks through  $U$ , while experts define success criteria via  $\mathcal{F}_{sc}$ . Conflicts between  $U$  and  $\mathcal{F}_{sc}$  can confuse  $\text{LLM}_R$ . To mitigate this, we employ another LLM ( $\text{LLM}_C$ ) to reconcile these potential discrepancies. Importantly,  $\text{LLM}_C$  does not share context with  $\text{LLM}_R$ , ensuring that the code of  $\mathcal{F}_{sc}$  is invisible to  $\text{LLM}_R$  [14].

**User description regularization.** Compared to  $\mathcal{F}_{sc}$ ,  $U$  can be more uncertain and flexible. To standardize  $U$ , we introduce a task-independent template  $T_U$ :

$$\mathbf{D}_U(U) \triangleq \text{LLM}_C(U, \mathcal{F}_{sc}, M \mid T_U) \quad (4)$$

After this prompting,  $\mathbf{D}_U(U)$  is expanded to include: (i) the composition of robots and objects in  $M$ . (ii) the goal states for these objects. (iii) the initial conditions, and (iv) potential post-goal states (see website for details on  $T_U$ ).

**Expert knowledge transfer.** When no success definition exists for a task in  $M$ ,  $\text{LLM}_C$  is further prompted to write the code for  $\mathcal{F}_{sc}$ :

$$[\mathbf{D}_U(U), \mathcal{F}_{sc}] \triangleq \text{LLM}_C(U, M \mid T_U, \mathbf{D}_U(U'), \mathcal{F}'_{sc}) \quad (5)$$

where  $(U', \mathcal{F}'_{sc})$  corresponds to another task defined on  $M$ . This paradigm increases the reusability of  $M$ , and also makes our framework less dependent on the existence of  $\mathcal{F}_{sc}$ .

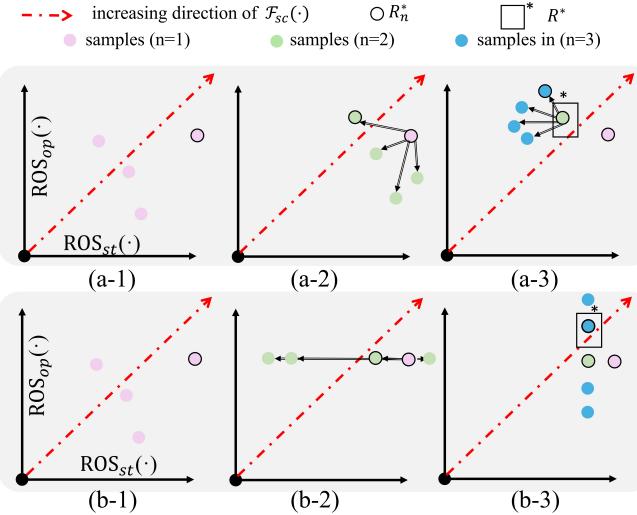


Fig. 3. Schematic illustration of the difference in sampling process for different on reward space  $\mathcal{R}$ . Compared to Eureka [14], observation Space disentanglement improves the efficiency of the sampling process of LLM by reducing the degrees of freedom.

### B. Observation Space Evaluation

In subsequent iterations,  $LLM_R$  is mainly guided by the following 3 types of guidance: (i)  $D_R(R_{n-1}^*)$  acts as the reward example. (ii)  $D_A[\Lambda(R_{n-1}^*)]$  reflects the detailed training effects of this example. (iii)  $D_G(\sum_m^{n-1} \mathcal{R}_m)$  keeps the history explore memory of  $LLM_R$ .

**Reward Observation Space.** We argue that the exploring ability of  $LLM_R$  iterations would be compromised when a code example from the last iteration are fed directly into the context of current iteration. Especially when  $R_{n-1}^*$  is mediocre, it would cause  $LLM_R$  to repeat this situation. To tackle this problem, we introduce the concept of *Reward Observation Space (ROS)*. This space contains a subset of all available environment-robot states  $ROS_{st}(R)$ , along with the operations defined upon those subset members  $ROS_{op}(R)$ . This make the information in a reward more compact and structured than line-by-line code.

**Observation Space disentanglement.** As shown in Fig. 3, we further disentangle the design of the **ROS** into two subproblems: space member selection  $ROS_{st}(R)$  and internal member operation  $ROS_{op}(R)$ . Consequently, the way  $D_R(\cdot)$  acts on  $R_n^*$  varies in different situations:

$$D_R(R_n^*) = \begin{cases} ROS_{st}(R_n^*), & \text{for odd } n \text{ or} \\ & \mathcal{F}_{sc}[\Lambda(R_n^*)] < \tau \\ ROS_{op}(R_n^*), & \text{otherwise} \end{cases} \quad (6)$$

The space member selection is triggered when the iteration index  $n$  is odd or the success score is less than a task-specific threshold  $\tau$ . In this situation,  $LLM_R$  is encouraged to select states from  $M$  that differ from  $ROS_{st}(R_n^*)$  as the observation members. On the other hand, the optimization to internal member operation is triggered when the success achieving by  $R_n^*$  is considerable within even iterations.  $LLM_R$  is prompted to use the same observation members in  $ROS_{st}(R_n^*)$  to devise novel reward items. This disentanglement strategy not only improves the efficiency of the reward design process but also

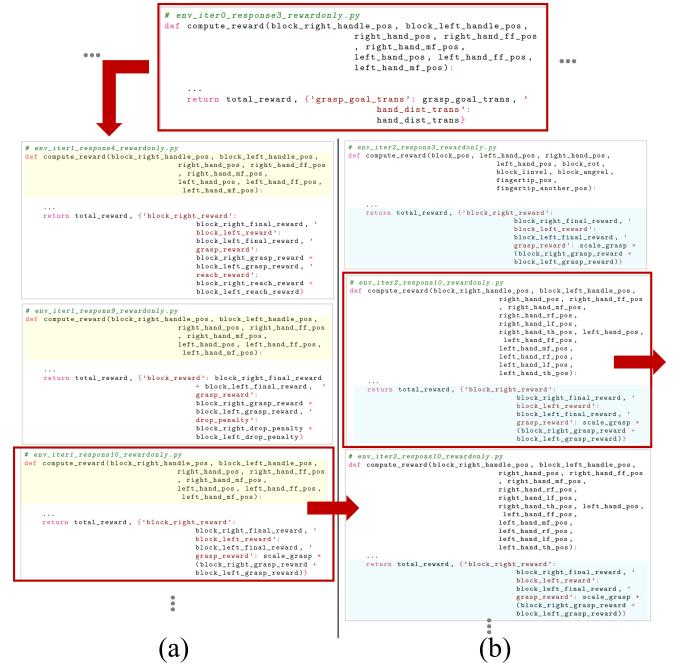


Fig. 4. Reward evolution in the first 3 iterations of our framework on **BlockGrasp** task. (a) Each sample in iteration 2 keeps the same  $ROS_{st}$  as  $R_1^*$ . (b) Each sample in iteration 3 keeps the similar  $ROS_{op}$  as  $R_2^*$ . In each iteration, the abstract part the highest  $\mathcal{F}_{sc}(\cdot)$  (identified by the red box) and two additional executable rewards are shown.

enhances the comparability of different **ROS** configurations within a single iteration, leading to more consistent and reliable reward outcomes.

**State execution table.** Considering the contradiction between the Markovian nature and the token consumption constrains of an LLM dialogue, most approaches [14], [3] keep LLM memory by resending part of the history as additional input. By contrast, we summarize the full history in a table form SET with less and constant token consumption. The column definition of SET is shown in Fig. : Column 1 enumerates the names of all states in  $M$ . Column 2 indicates how many times each state is adopted in all historical rewards. Column 3 indicates the contribution to the task success of each state. This is calculated by dividing  $\mathcal{F}_{sc}[\Lambda(R)]$  evenly to each state contained in  $ROS_{st}(R)$ . The information on table is accumulated along the evaluation:

$$D_G\left(\sum_n^n \{R_n^{(k)}\}_{k=1}^K\right) \triangleq \text{SET}_{n-1} \oplus \sum_k^K ROS_{st}(R_n^{(k)}) \quad (7)$$

Since a failed run  $R$  implies that the corresponding  $ROS_{st}(R)$  has not efficiently explored, it is not accumulated on SET.

### C. Implementation details

**LLM settings.** We use GPT-4, in particular the gpt-4-0314 variant, as the backbone API for both  $LLM_R$  and  $LLM_C$ . Besides passing  $D_U(U)$ , the two LLMs do not share context with each other. During prompting, the temperature is set as 1.0 to keep the diversity of reward samples. Each evolution is set to  $N = 5$  iterations, and LLM is required to generate  $K = 16$  samples simultaneously in each  $LLM_R$  design iteration.  $ROS_{st}$  is implemented by retaining only the first and last

logical line of the reward code.  $\tau$  reflects the difficulty of the given task. The success threshold  $\tau$  for each task is pre-set as the average success rate of  $K_0 = 16$  independent and executable rewards designed by  $LLM_R$ . All prompts and task-independent templates are attached to our website.

**RL settings.** All environment code  $M$  are deployed as IsaacGym [16] environments, which can be simulated simultaneously with high efficiency. Proximal Policy Optimization (PPO) algorithm [22], which uses default training parameters in the environments [7], is used as a practice for RL algorithms  $\Lambda$ . The rewards designed in the same iteration are distributed and trained on 8 NVIDIA RTX 3090 GPUs. The optimal reward candidate  $R_N^*$  are evaluated in 5 independent environments. The maximum interaction epochs per  $\Lambda$  is set to 3000 in each  $LLM_R$  design process, and 6000 in each evaluating process.

## V. EXPERIMENT

### A. Baselines and Tasks

**L2R** [32] is a non-iterative framework that relies on reward function templates. For environments and tasks specified in natural language, the first LLM was asked to fill in a natural language template describing the agent’s movements; the second LLM was then asked to convert this “action description” into code that invoked a set of manually-defined reward API primitives to write a parameterized reward program. To make the L2R competitive to our work, we follow [14] to define motion description templates to mimic the original L2R templates. Its API reward primitives were constructed using individual components of the original human rewards.

**Eureka** [14] utilizes the code-writing and zero-sample-generation capabilities of Large Language Models (LLMs) to generate executable reward function code directly from the environment source code and linguistic task descriptions in an iterative framework. This process does not require task-specific hints, enabling Eureka to achieve zero-sample reward function generation across a wide range of tasks. Eureka employs an evolutionary search strategy by iteratively sampling reward functions, evaluating their performance, and incrementally improving the reward functions based on feedback from these evaluations. For the fairness of the comparison, our approach is consistent with Eureka in terms of iteration amount  $N = 5$  and sample amount  $K = 16$ .

The above baselines and our framework are evaluated on 20 hand-object interaction task from Bi-dexterous Manipulation benchmark [7] with the same hyperparameter settings. These task can be further categorized as: (i) Block Manipulation ( $\times 3$ ): BlockPush, BlockStack, Grasp&Place. (ii) Door Manipulation ( $\times 4$ ): DoorCloseOutward, DoorCloseInward, DoorOpenOutward, DoorOpenInward. (iii) In-hand Manipulation ( $\times 6$ ): ReOrientation, HandOver, CatchUnderarm, CatchOver2Underarm, CatchAbreast, TwoCatchUnderarm. (iv) Functional Grasp ( $\times 8$ ): Pen, Switch, GraspAndPlace, Kettle, Scissors, SwingCup, BottleCap, LiftUnderarm. It is noted that there are no templates or modules specifically designed for the Bi-dexterous Manipulation in our framework.

The 9 tasks in IsaacGymEnvs [16] are ignored in our experiments, mainly stemming from the fact that they contain relatively few environment states and that Eureka can already design better forms of rewards.

### B. Evaluation Metrics

The following metrics are used to quantitatively evaluate the quality of the different reward design approaches in each task. It is worth noting that the quantities between different tasks are relatively independent and not comparable.

**Evaluated success rate (ESR)** for each task indicates the maximum success achieved in once RL training procedure with a given reward. Consistent with Eureka [14], 5 identical but independent RL training procedures are launched after each reward design procedure. The average of the 5 rates ( $ESR_{avg}$ ) is adopted to measure the effective of the given reward and a higher  $ESR_{avg}$  means a more effective reward. **Sampling state disparity (SSD)** for each task refers to the bias between the maximum and average usage frequency of all states for historical reward codes (whether successfully executed or not). It reflects the exploration degree of a reward design approach on all states within the task.

### C. Comparisons

Fig. 5 reports the performance comparison of our framework with other existing approaches on 20 tasks.  $ESR_{avg}$  is used as the evaluation metric. According to the results in the table, our method performs better than Eureka in 9 tasks and matches Eureka’s performance in 5 extreme simple/difficult tasks. Only in 6 tasks (BlockPush, CatchAbreast, CatchOver2Underarm, DoorCloseOutward, ReOrientation, LiftUnderarm), our method performs slightly lower than Eureka.

We further found that it was the success rate thresholds for these tasks that were set too high, causing  $LLM_R$  to focus too much on space member selection throughout the evolution process. In our ablation study, two underperforming tasks (CatchAbreast and DoorCloseOutward) as well as two outperforming tasks are selected. However, in  $2+D_R(\cdot)^-$  where the threshold is set to 0.1, we outperformed Eureka on both underperforming tasks, achieving scores of [0.54, 1], which are higher than Eureka’s scores of [0.5, 0.96]. This suggests that adjusting thresholds might have a significant impact on improving learning efficiency and task performance across various robotic tasks. In our current approach, we use the average of the success rates achieved by LLM in the first iteration of the reward design as the average success rate for each task. In the future, an adaptive threshold search algorithm could enable this mechanism to achieve better performance.

### D. Ablation Study

Starting with the Baseline+ $D_U(\cdot)$ , we observe that the naive introduction of the user-expert mission reconciling mechanism can marginally improve LLM reward design. This means that only formal and informative requirements descriptions are not sufficient for LLM based reward evolution. When the state execution table (SET) is introduced (1+SET), we observe a more consistent improvement, especially in

TABLE I

**ABLATION STUDIES ON 4 BI-DEXTEROUS MANIPULATION TASKS.** FROM TOP TO BOTTOM, EACH ROW REPORTS A VARIANT FRAMEWORK THAT MAKES ONLY ONE MODIFICATION W.R.T. OUR FINAL VERSION (THE LAST ROW) AS SPECIFIED BY ITS NAME.

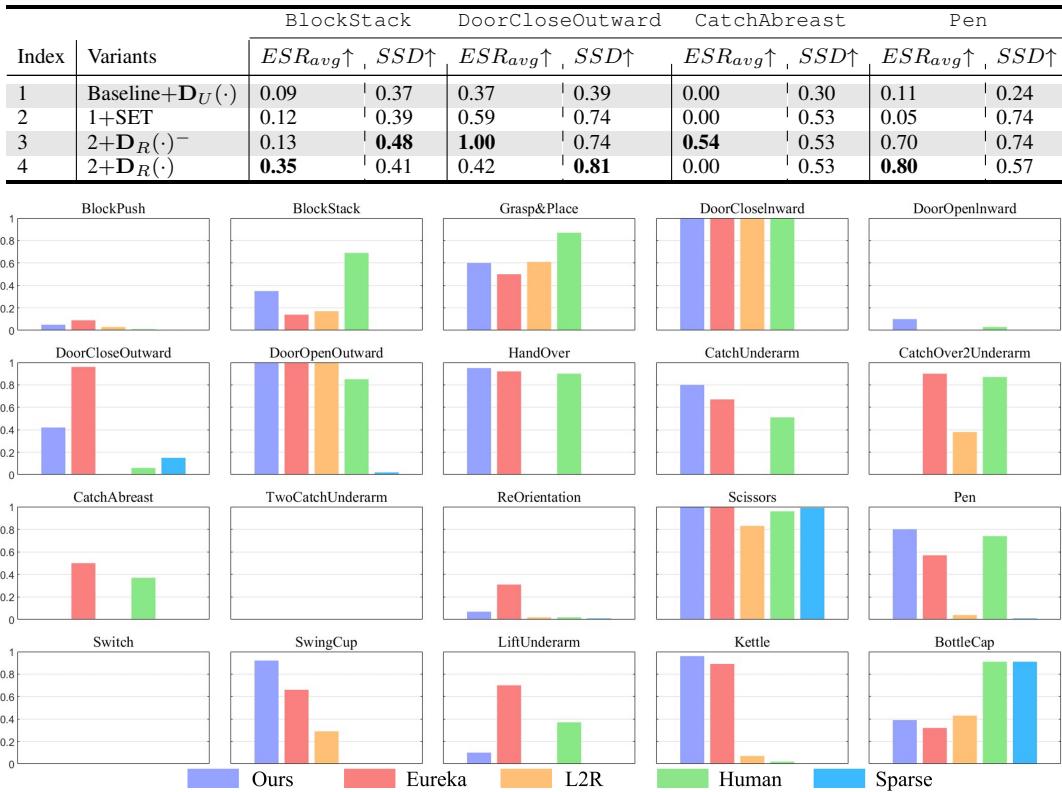


Fig. 5. **Comparison with existing LLM reward design approaches.** The subgraphs report the success rates on the 20 dexterity tasks on the Bi-dexterous Manipulation benchmark [7].

the *DoorCloseOutward* task where  $ESR_{avg}$  increases from 0.37 to 0.59, and  $SSD$  improves significantly. This suggests that incorporating structured state execution improves the system’s ability to handle more complex scenarios by providing a clearer execution plan. The addition of a fixed success threshold  $\tau = 0.1$  in variant (iii),  $2+\mathbf{D}_R(\cdot)^-$ , further boosts performance, particularly in tasks like *BlockStack* and *CatchAbreast*, where  $ESR_{avg}$  and  $SSD$  either match or exceed prior results. The fixed threshold likely provides a clearer metric for success, leading to more stable training and execution. Finally, variant (iv),  $2+\mathbf{D}_R(\cdot)$ , which introduces a task-specific success threshold via automatic hyperparameter search, results in significant improvements in tasks like *BlockStack* (with an  $ESR_{avg}$  of 0.35) and *Pen* (with an  $ESR_{avg}$  of 0.80). However, the improvement is not uniform across all tasks, as the *CatchAbreast* task continues to show little progress, indicating that task-specific thresholds may not universally enhance performance across all environments.

## VI. CONCLUSION

In this work, we introduced a novel framework that enhances the design of reinforcement learning (RL) rewards by harnessing the capabilities of large language models (LLMs). Our approach addresses the critical challenge of grounding, facilitating more effective communication between LLMs and RL agents by evolving the Reward Observation

Space (**ROS**) through heuristic sampling. By incorporating a table-based exploration caching mechanism, we alleviate the Markovian constraint commonly found in LLM dialogues, enabling a more comprehensive and efficient exploration of potential reward spaces. Additionally, our structured text-code reconciliation strategy bridges the cognitive gap between user intentions and expert-defined success criteria, ensuring that LLM-generated rewards are both relevant and actionable across a variety of environments. The proposed method significantly surpasses existing frameworks in both efficiency and effectiveness, highlighting its potential for universal RL reward design. This research represents a notable advancement in integrating LLMs with RL, paving the way for the development of autonomous systems that can adapt to diverse tasks with minimal human input. Future work will aim to further refine the feedback loop and investigate the application of this framework in more complex, real-world scenarios.

## REFERENCES

- [1] Llms don't understand negation. Available from: <https://hackernoon.com/llms-dont-understand-negation>. Accessed: 2024-08-22.
- [2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i

- say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- [4] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik’s cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.
  - [5] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.
  - [6] Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. Rt-h: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024.
  - [7] Yuanpei Chen, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen McAleer, Hao Dong, Song-Chun Zhu, and Yaodong Yang. Towards human-level bimanual dexterous manipulation with reinforcement learning. *Advances in Neural Information Processing Systems*, 35:5150–5163, 2022.
  - [8] Peter Dayan and Bernard W Balleine. Reward, motivation, and reinforcement learning. *Neuron*, 36(2):285–298, 2002.
  - [9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
  - [10] Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conf. Robo. Learn.*, pages 3766–3777. PMLR, 2023.
  - [11] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.
  - [12] Joanne Taery Kim and Sehoon Ha. Observation space matters: Benchmark and optimization algorithm. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1527–1534. IEEE, 2021.
  - [13] Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.
  - [14] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
  - [15] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. *arXiv preprint arXiv:2406.01967*, 2024.
  - [16] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.
  - [17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
  - [18] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.
  - [19] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.
  - [20] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.
  - [21] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
  - [22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
  - [23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
  - [24] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.
  - [25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
  - [26] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.
  - [27] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.
  - [28] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839*, 2023.
  - [29] Zeqi Xiao, Tai Wang, Jingbo Wang, Jinkun Cao, Wenwei Zhang, Bo Dai, Dahua Lin, and Jiangmiao Pang. Unified human-scene interaction via prompted chain-of-contacts. *arXiv preprint arXiv:2309.07918*, 2023.
  - [30] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.
  - [31] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.
  - [32] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. In *7th Annual Conference on Robot Learning*, 2023.
  - [33] Yuwei Zeng, Yao Mu, and Lin Shao. Learning reward for robot skills using large language models via self-alignment. *arXiv preprint arXiv:2405.07162*, 2024.
  - [34] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.