R^2E: A Role-driven Reward Evolutionary Framework for Automated Reward Function Design

Shouhao Chang[1] **Xuan Liu**[1*] **Hongye Zhu**[1] **Xinning Chen**[1] **Shigeng Zhang**[2*]
[1]Hunan University
[2]Central South University

Abstract

Designing effective reward functions is a critical challenge in Reinforcement Learning (RL), which traditionally requires costly manual trial-and-error. While Large Language Model (LLM)-based methods have shown promise in reward design, undifferentiated sampling and limited use of historical feedback often lead to insufficient exploration of the reward space and optimization instability. To address these challenges, we propose **R^2E**, a **R**ole-driven **R**eward **E**volutionary framework for automated reward function design that structures the process through explicit role specialization. R^2E employs multiple LLM roles with complementary objectives: an Explorer that promotes novelty to expand the search space, a Guardian that performs conservative refinements to improve training stability, and an Evolver that recombines high-performing rewards via crossover and mutation to integrate effective structures. A dynamic scheduling strategy coordinates these roles, progressively shifting the search from broad exploration to focused exploitation. To mitigate optimization instability and provide a medium for cross-role collaboration, R^2E incorporates a Global Elite Pool that retains the best-performing rewards to guide subsequent generations, thereby enabling coordinated interaction among roles and ensuring a reliable and consistent refinement process. Extensive experiments on multiple robotic tasks demonstrate the effectiveness of our method. Code is available at https://github.com/ChangYuandao/R2E.

1 Introduction

Reward function design is a fundamental component in Reinforcement Learning, directly shaping agent behaviors and training stability [26, 22], especially in complex robotic control tasks [15, 25]. Although the field of robotic control has witnessed significant progress [14, 3, 6], achieving high-performance behaviors often necessitates intricate reward shaping. However, manual reward engineering remains heavily dependent on domain expertise [5], which significantly limits its scalability across diverse tasks. To address these challenges, Inverse Reinforcement Learning (IRL) [21, 31] and Preference-based Reinforcement Learning (PbRL) [27, 9] have emerged as alternatives to reduce expert reliance. Nevertheless, they typically incur prohibitive data costs in physical environments [4] and yield black-box representations that lack the structural transparency essential for safety and debugging [23].

Recent advances in LLM have demonstrated strong capabilities in code synthesis and logical reasoning [2, 7], enabling new approaches to reward function design [29, 28]. Building on these capabilities, prior work has proposed iterative frameworks that alternate between reward generation, policy training, and refinement based on feedback [17]. By leveraging environmental feedback, these instruction-driven frameworks enable automated reward discovery, demonstrating substantial potential for scaling to complex, real-world RL applications [18].

*Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

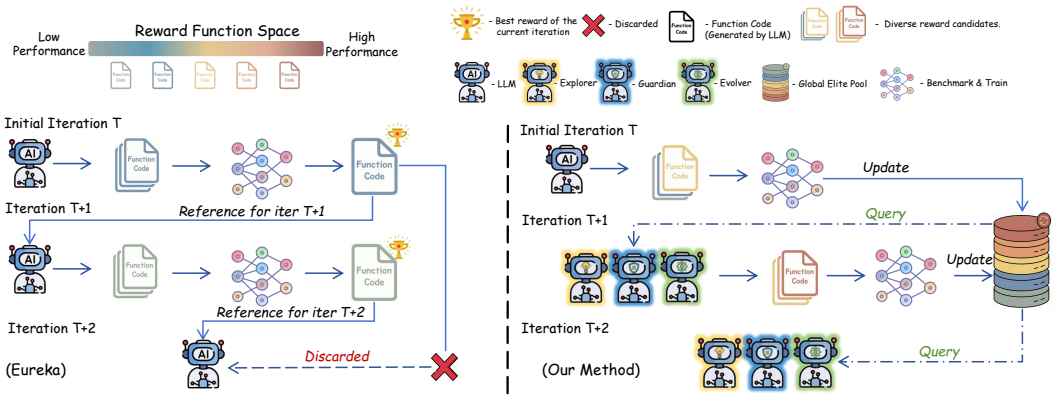


Figure 1: **Comparison of reward design paradigms. (Eureka)** generates homogeneous rewards and lacks memory. If iteration $T+1$ regresses, the LLM references only $T+1$ for $T+2$, discarding high-performing rewards from T. **(Our Method)** populates a Global Elite Pool via initial exploration. Specialized roles query historical elites from the pool to iteratively evolve diverse reward candidates.

Despite these advances, existing LLM-based reward generation frameworks exemplified by Eureka are still constrained by two key limitations, as illustrated in Fig. 1(Left). First, reward candidates are typically produced through undifferentiated sampling under a monolithic prompting paradigm, where diversity arises mainly from stochasticity rather than a structured search process, making it difficult to explicitly balance exploration and exploitation. Second, most iterative frameworks rely solely on the best reward and feedback from the immediate preceding iteration, lacking a persistent mechanism to preserve and reuse historically high-performing rewards. As a result, due to the stochasticity in LLM generation, the candidates produced in the next iteration may be inferior to those from the previous iteration, leading to unstable optimization and potential performance regression.

To this end, we propose **R^{2}E**, a role-driven reward evolutionary framework for automated reward function design that structures the process through explicit role specialization, as illustrated in Fig. 1(Right). R^{2}E structures the reward search process into three complementary roles: the Explorer, the Guardian, and the Evolver. Through a dynamic scheduling strategy, the framework adaptively adjusts the sampling allocation of each role across iterations, enabling a gradual transition from broad exploration to stable refinement. Furthermore, R^{2}E incorporates a Global Elite Pool as a persistent retention mechanism to store globally best-performing reward functions. This pool provides high-quality references for subsequent iterations, enabling best-so-far rewards retention and mitigating optimization instability in iterative LLM-based optimization.

The primary contributions of this work are summarized as follows:

- We propose R^{2}E, a structured reward evolutionary framework that leverages explicit role specialization and a dynamic scheduling strategy to guide reward function design, enabling a progressive shift from broad exploration to focused exploitation.
- To facilitate cross-role collaboration, we introduce a Global Elite Pool. By actively retaining the highest-performing reward functions, it mitigates optimization instability and provides a stable foundation for roles to collectively refine designs.
- We demonstrate that R^{2}E consistently outperforms baselines across 14 tasks from Isaac-Gym and Bi-DexHands by effectively discovering high-performing reward functions for locomotion and manipulation.

2 Related Work

Large Language Model for Reward Engineering. Departing from manual engineering and data-intensive Inverse RL [1], recent research has pioneered the use of LLM for automated reward design [29, 28, 17]. Early frameworks such as L2R [29] and Text2Reward [28] established the

potential of LLM but remained heavily reliant on task-specific templates, as well as few-shot prompting and human feedback.

LLM-driven Evolutionary Reward Design. Recent studies adopt an evolutionary paradigm to refine rewards via the iterative potential of LLM. A landmark work in this category is Eureka [17], which introduced a fully autonomous iteration cycle consisting of reward generation, training, and reflection. By utilizing environmental feedback to refine rewards without human intervention, Eureka demonstrated that LLM could independently discover effective reward structures. Subsequent research [13, 30, 16, 12, 10] has integrated additional strategies to enhance this evolutionary process. For example, Revolve [12] evaluates trajectories using an ELO-based scoring system, though it still incorporates costly human feedback. RF-Agent [10] regards the LLM-based reward function design as a Monte Carlo Tree Search process. Critically, these methods (including Eureka) treat LLM as a role-agnostic stochastic sampler, which can hinder effective exploration and exploitation of high-performing outcomes across iterations. Building upon evolutionary frameworks like Eureka, R^2E replaces undifferentiated sampling with parallel, role-driven specialization to iteratively optimize rewards, achieving a principled balance between exploration diversity and stable performance.

3 Problem Setting

We study automated reward design for reinforcement learning in complex environments (e.g., physics simulations). Similar to meta-reinforcement learning [11], we formulate this as a bi-level optimization problem: an outer loop searches a structured reward space generated by an LLM, while an inner-loop RL algorithm learns a policy under a fixed reward. A task is modeled as an MDP [24] $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $\mathcal{P}$ the transition dynamics, R the reward function, and $\gamma \in [0, 1]$ the discount factor. In our framework, rewards are not predefined but generated by an LLM from a structured space $\mathcal{R}$, with $r \in \mathcal{R}$ denoting a candidate reward.

Given a reward function r, the corresponding policy is obtained by solving the inner-loop optimization:

$$\pi_r = \arg\max_{\pi} \mathbb{E}_{\tau \sim (\pi, \mathcal{P})} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{1}$$

where $\tau = (s_0, a_0, \ldots, s_T)$ denotes a trajectory generated by executing policy π in the environment. Through this process, the agent interacts with the environment's physics $\mathcal{P}$ and learns to master complex skills, using the reward function r as a guide for its behavioral improvement. The objective of reward design is to identify an optimal reward function in the outer loop:

$$r^* = \arg\max_{r \in \mathcal{R}} \mathcal{J}(\pi_r) \tag{2}$$

where $\mathcal{J}(\pi_r)$ denotes the expected task-level performance achieved by the policy π_r trained under reward r. On IsaacGym, $\mathcal{J}$ is a task-specific performance metric; on Bi-DexHands, it is the success rate, defined on $[0, 1]$. This task performance metric serves as a global evaluation of the reward's effectiveness in guiding the agent toward the successful completion of the target task. This metric is used exclusively for outer-loop reward selection and is not directly observable or optimized by the inner-loop RL algorithm.

To solve this outer-loop optimization, we adopt an iterative reward evolution process. At each iteration t, the framework maintains a fixed-capacity Global Elite Pool $\mathcal{B}_t = \{r^{(1)}, \ldots, r^{(K)}\}$ that serves as a repository for the most effective reward functions evaluated throughout the evolution. Conditioned on the historical rewards in $\mathcal{B}_t$, explicit role specialization is employed where multiple specialized roles including Explorer, Guardian, and Evolver collaboratively produce a set of candidate rewards:

$$\mathcal{R}_t = \mathcal{R}_t^{\text{EXPLORER}} \cup \mathcal{R}_t^{\text{GUARDIAN}} \cup \mathcal{R}_t^{\text{EVOLVER}} \tag{3}$$

characterized by complementary search objectives and dynamically adjusted sample budgets to facilitate a progressive shift from broad exploration to focused exploitation. Each candidate reward $r \in \mathcal{R}_t$ is evaluated through RL training on IsaacGym or Bi-DexHands tasks. If its performance $\mathcal{J}(\pi_r)$ exceeds that of the worst-performing reward in the current Global Elite Pool, it replaces that inferior candidate, ensuring the pool's quality consistently improves until the update of $\mathcal{B}_{t+1}$ is complete. After a predefined number of iterations, the final reward is selected as the best-performing element in the Global Elite Pool.

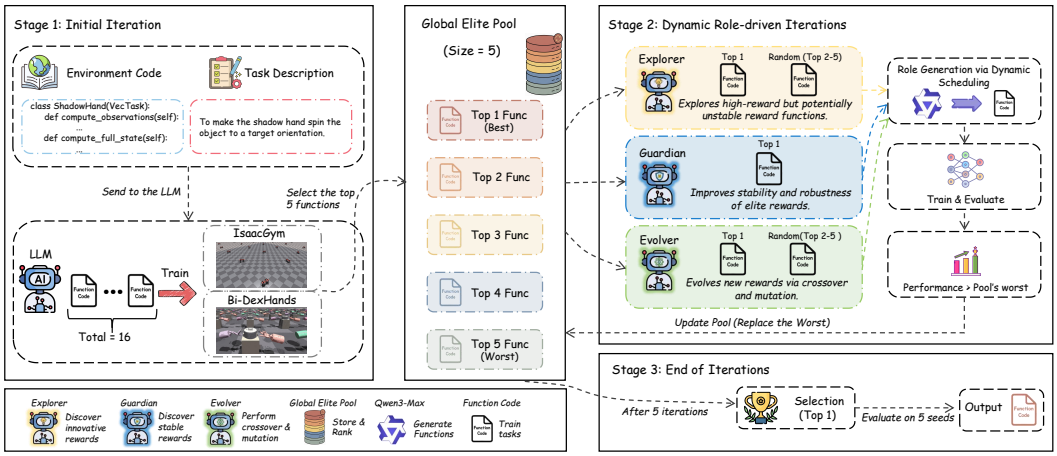


Figure 2: **Overall architecture of R^{2}E.** The framework operates in three main stages: **(Stage 1 Initial Iteration)**: Reward candidates are generated through a broad exploration phase based on the environment code and task descriptions, and then trained on IsaacGym or Bi-DexHands tasks to initialize the Global Elite Pool with the Top 5 functions. **(Stage 2 Dynamic Role-driven Iterations)**: In each iteration, a total of $N = 16$ candidates are generated through dynamic scheduling by the Explorer, Guardian, and Evolver roles. After RL training and evaluation, the pool is dynamically updated by replacing the worst-performing reward. **(Stage 3 End of Iterations)**: After the search concludes, the Top 1 reward is selected and re-evaluated across five random seeds to produce the final output. **(Global Elite Pool)**: A fixed-capacity memory ($K = 5$) that stores and ranks the best-performing reward functions across iterations.

4 Method

In this section, we present **R^{2}E**, a role-driven evolutionary framework that combines multi-role collaborative reward design with a Global Elite Pool to ensure both stability and diversity in the reward search process.

4.1 Overview

R^{2}E implements a stage-wise evolutionary strategy that begins with an initial exploration phase to ensure broad coverage of the reward design space before entering role-driven evolution, as illustrated in Fig. 2.

In the first iteration, reward candidates are generated in an initial exploration phase to avoid prematurely restricting the search to narrow subspaces. This unconstrained initialization establishes sufficient structural diversity, providing a robust foundation for subsequent role-driven evolution.

Starting from the second iteration, the framework enters a role-driven phase in which the Explorer, Guardian, and Evolver collaboratively generate reward candidates conditioned on the Global Elite Pool. Guided by a dynamic scheduling strategy, this phase balances exploration, refinement, and the inheritance of effective reward structures.

After the evolutionary process terminates, the framework selects the top-ranked reward function from the Global Elite Pool as the final output of R^{2}E and re-evaluates it across multiple random seeds for robust performance estimation.

4.2 Role-driven Reward Evolutionary Design

Building upon the initial seed population, R^{2}E introduces three specialized roles to steer the evolutionary process. These roles operate in a coordinated manner by querying the Global Elite Pool, enabling the framework to balance exploration, conservative refinement, and the inheritance of effective reward structures. Detailed design specifications are provided in Appendix A.

Explorer. The Explorer is tasked with diversifying the reward function space by prioritizing structural creativity and unconventional reward formulations, with limited emphasis on immediate training stability. Specifically, the Explorer queries two reward functions from the Global Elite Pool, namely the current top-performing reward and a randomly sampled remaining reward, to serve as behavioral anchors. By analyzing the specific behaviors encouraged or overlooked by these references, the Explorer deliberately proposes new reward terms and behavioral logic that diverge from established patterns. This role focuses on exploring diverse reward structures and innovative shaping terms while strictly adhering to physical plausibility and numerical safety constraints. By explicitly shifting its heuristic focus away from existing elite reward functions, the Explorer prevents the search from stagnating in local optima and uncovers novel behavioral patterns that facilitate higher performance ceilings.

Guardian. The Guardian focuses on enhancing the robustness and stability of reward functions through conservative refinement. Specifically, the Guardian queries the current best reward function from the Global Elite Pool to analyze its behavioral metrics and identify potential issues such as action oscillations or component imbalances. By applying stabilizing operations, including smoothing reward scales and rescaling coefficients, the Guardian generates improved variants that promote steady and predictable performance gains. These refinements are implemented to ensure numerical stability and magnitude balancing while preserving the core task logic of the original reward function. Through this local optimization process, the Guardian discovers more trainable versions of established rewards and prevents performance regressions caused by noisy or unstable reward signals.

Evolver. The Evolver crosses over and mutates high-performing reward functions to synthesize superior offspring with enhanced behavioral characteristics. Specifically, the Evolver queries the current top-performing reward function and a randomly sampled candidate from the remaining members of the Global Elite Pool to serve as parent formulations. It identifies and merges complementary components between these diverse parents, such as combining terms that facilitate smooth motion from one with those that drive high task success from another. Simultaneously, the Evolver introduces controlled variations through weight recalibration, shaping adjustments, or the addition of safety constraints. By balancing the inheritance of successful traits with the introduction of innovative variations, the Evolver ensures that new formulations remain grounded in prior designs while promoting continual improvement.

4.3 Global Elite Pool

We maintain a Global Elite Pool $\mathcal{B} = \{r_1, r_2, \ldots, r_K\}$ as a size-constrained repository of the best-performing reward functions discovered during the evolutionary process. Each $r_i \in \mathcal{B}$ represents a reward function admitted after performance evaluation. By setting $K = 5$, we preserve diverse behavioral references while excluding lower-performing candidates that might degrade the search. This pool acts as a persistent memory, ensuring that successful reward structures are retained across iterations despite the inherent stochasticity of LLM generation.

Competitive Admission Strategy. Each candidate reward function r produced by the LLM is evaluated by $\mathcal{J}(\pi_r)$ within the target environment: a task-specific performance metric on IsaacGym, and the success rate on Bi-DexHands. To ensure a monotonically improving boundary of reward quality, we implement a competitive admission criterion. A new reward function r is admitted into $\mathcal{B}$ if and only if $\mathcal{J}(\pi_r) > \min_{r_i \in \mathcal{B}} \mathcal{J}(\pi_{r_i})$. Upon admission, the reward function with the lowest task performance is evicted to maintain the fixed capacity. This strategy transforms the pool into a dynamic filter, establishing a consistently rising performance floor that stabilizes the evolutionary trajectory.

Inter-Role Synergy via Shared Memory. Beyond simple storage, the Global Elite Pool functions as a central hub that facilitates synergy and interconnection among the specialized generation roles. By querying elite reward functions from the pool, the Explorer, Guardian, and Evolver transition from isolated zero-shot generation to a collaborative, guided in-context evolution. This shared memory provides the LLM with concrete examples of best-performing reward functions and their corresponding performance metrics, serving as a structured prior that provides guidance for the search processes of different roles. By grounding their specific heuristics in proven behavioral patterns, the roles effectively synchronize their efforts to uncover higher performance ceilings.

Table 1: **Performance Comparison on IsaacGym Tasks.** All results report task-specific performance metrics as mean ± standard deviation across runs. **Bold** indicates the best results within each group. Avg norm score denotes the average human-normalized performance computed as $(\text{Method} - \text{Sparse})/(\text{Human} - \text{Sparse})$ for each task.

Method	Ant	Anymal	Quadcopter	AllegroHand	FrankaCabinet	ShadowHand	Avg Norm Score
Sparse	1.0000±0.0000	-1.9619±0.2734	-0.3447±0.2361	0.0619±0.0415	0.0000±0.0001	0.0546±0.0046	0.0000
Human	6.3275±0.3159	-0.0232±0.0030	-0.0906±0.0095	11.3698±0.8865	0.1230±0.0892	6.9433±1.3385	1.0000
Eureka	6.5876±0.2602	-0.0249±0.0019	-0.0415±0.0114	19.2377±3.0125	0.3816±0.2243	4.9272±2.3201	1.4578
REvolve	4.4297±1.1706	-0.0172±0.0021	-0.0388±0.0135	21.9641±1.3215	0.7224±0.152	8.1640±1.6612	1.8500
RF-Agent	6.7225±0.2767	-0.0154±0.0053	-0.0274±0.0028	20.4162±2.1523	0.8156±0.1571	7.9717±3.5775	2.1512
Ours	**7.2639**±0.4584	**-0.0135**±0.0043	**-0.0211**±0.0076	**23.4458**±0.7224	**1.0000**±0.0000	**8.9603**±1.4171	**2.4908**

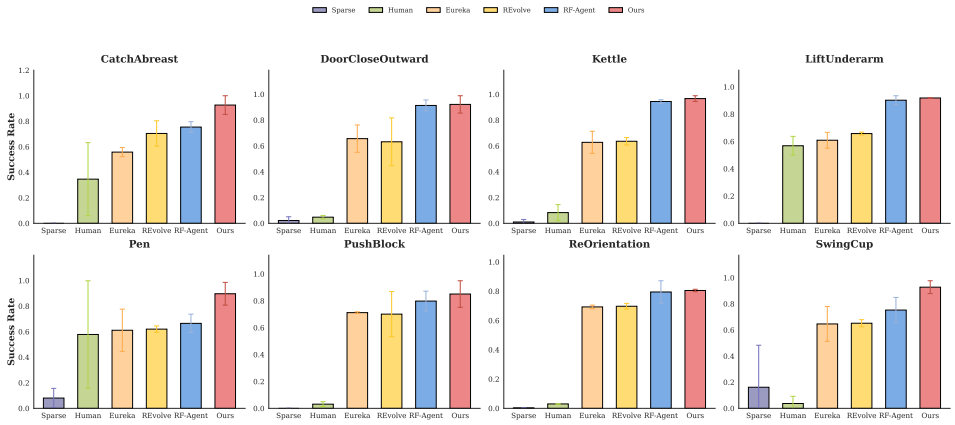


Figure 3: **Performance Comparison on Bi-DexHands Tasks.** Bars and error bars denote the mean and standard deviation of success rates over 5 random seeds, respectively. R^2E achieves the highest mean success rate and maintains competitive stability across all environments.

4.4 Dynamic Scheduling Strategy

We employ a dynamic scheduling strategy to manage the transition from exploration to refinement by reallocating a fixed budget of $N = 16$ candidates per iteration. Starting from the second iteration, this strategy distributes the budget across roles to guide their search based on the Global Elite Pool. We initially prioritize the Explorer to maximize coverage and novelty. As the search nears the limit T, the allocation shifts toward the Guardian and Evolver to transition from discovery to stability and recombination. This adaptive scheduling gradually shifts the focus from broad exploration to conservative refinement and inheritance, ensuring both diverse search coverage and deep optimization of reward functions. Detailed formal definitions and the specific scheduling formulas for this reallocation are provided in Appendix D.2.

5 Experiments

5.1 Experimental Setup

We evaluate our framework on 14 diverse tasks from IsaacGym [20] and Bi-DexHands [8], including six representative locomotion and manipulation tasks in IsaacGym and eight complex Bi-DexHands tasks that require sophisticated dual-arm coordination,additional environment descriptions are provided in Appendix C. Evaluation criteria are adapted to each benchmark: Bi-DexHands uses success rate because task completion is determined by a binary signal of zero or one, while IsaacGym uses task-specific performance metrics (e.g., forward speed for *Ant*). We employ *Qwen3-Max* as the LLM backbone for both our method and all baselines to ensure a fair comparison and reliable role execution during reward design. Further implementation details are provided in Appendix D.

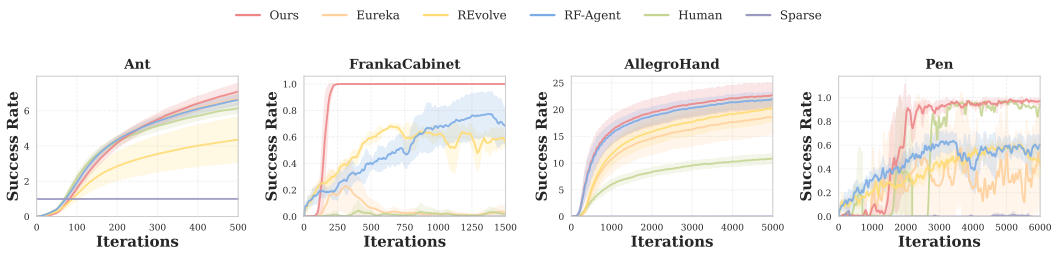


Figure 4: Training curves on representative locomotion and manipulation tasks from IsaacGym and Bi-DexHands. Solid lines and shaded regions denote the mean and standard deviation over 5 seeds. IsaacGym curves report task-specific performance metrics, and Bi-DexHands curves report success rate.

5.2 Baselines

Sparse replaces the dense reward with a task-level signal that is provided only when the task is successfully completed. As a result, the reward is extremely sparse, which makes reinforcement learning significantly more challenging.

Human uses the dense reward functions provided by the original IsaacGym and Bi-DexHands benchmarks. These rewards were written by the researchers who designed the environments and therefore represent expert-crafted reward engineering.

Eureka [17] automatically generates dense reward functions by leveraging natural language task descriptions and environment observations. The method produces reward functions in batches and iteratively selects the top-performing candidate for subsequent refinement.

REvolve [12] utilizes a population-based approach, employing evolutionary operators to refine reward functions based on human feedback. To align with our objective of fully autonomous reward design, we utilized the auto version of REvolve, substituting human-in-the-loop input with automated environmental feedback.

RF-Agent [10] is an automated reward function design framework that combines Large Language Models with Monte Carlo Tree Search, viewing reward design as a sequential decision-making process.

5.3 Main Results

We first evaluate on six representative tasks from IsaacGym, covering both locomotion tasks and manipulation tasks. As shown in Table 1, R^2E consistently achieves the best overall performance across all tasks compared with the considered baselines. The improvements are especially clear on manipulation tasks, demonstrating the capability of R^2E to handle more complex tasks characterized by high-dimensional environments and complex dynamics.

As illustrated in Figure 3, R^2E achieves superior performance on the Bi-DexHands benchmark. We observe that our method consistently surpasses baselines, with a substantial margin in challenging tasks like *CatchAbreast* and *SwingCup*. Collectively, these results indicate that effective reward functions can be automatically designed even in more complex task environments, validating the efficacy of our method.

To provide a more granular analysis of the training dynamics, Figure 4 depicts the learning curves for four representative tasks covering both locomotion and manipulation. R^2E exhibits superior performance compared to baselines. Notably, in the *FrankaCabinet* task, R^2E achieves rapid convergence and effectively escapes local optima to learn a viable policy. Furthermore, in the more complex *Pen* task, R^2E maintains high success rates with lower variance in the latter stages of training. These experimental results demonstrate the capability of our method to design robust reward functions that facilitate stable and reliable policy learning.

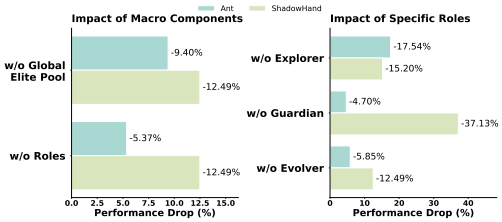


Figure 5: Ablation study on *Ant* and *ShadowHand* tasks. The charts display the performance drop (%) relative to the full method. **Left:** Ablation of macro components. **Right:** Ablation of specific roles. The length of the bars indicates the importance of each component.

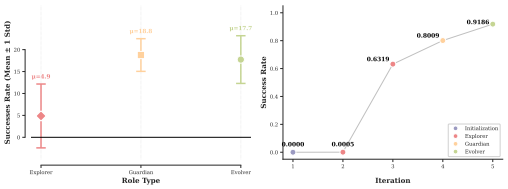


Figure 6: Analysis of Role Distinctiveness and Synergistic Evolution. **Left:** Validates the statistical characteristics of the roles, where μ denotes the mean reward. **Right:** Illustrates the synergistic mechanism in a representative evolutionary trajectory.

5.4 Ablation Studies

To understand the contribution of each component in our framework, we conduct ablation experiments on *Ant* (locomotion) and *ShadowHand* (manipulation). The results are illustrated in Figure 5, with a detailed analysis provided below.

w/o Roles & Global Elite Pool. Removing role specialization causes clear performance degradation, indicating that role-driven diversity effectively expands the search space. Similarly, ablating the Global Elite Pool by restricting reference functions to only the best result from the previous iteration severely destabilizes optimization. Together, these results show that both diversified generation patterns and the preservation of globally superior references are essential for discovering high-performing reward functions.

w/o Individual Roles. To evaluate the contribution of each role, we conducted ablation experiments by removing the Explorer, Guardian, or Evolver and redistributing its responsibilities among the remaining modules. The results show that the removal of any single role leads to significant performance degradation. Notably, the removal of the Guardian causes a substantial drop in performance on the *ShadowHand* task, reflecting the difficulty of reward design in complex tasks and highlighting the Guardian's critical role in stabilizing the learning process. These findings demonstrate that each module is indispensable in the iterative optimization process, as they collectively provide complementary mechanisms of exploration, refinement, and evolution to sustain continuous improvement.

Impact of Dynamic Scheduling Strategy. We evaluate scheduling effectiveness by comparing our Dynamic Scheduling Strategy with a fixed baseline that uses 5 Explorers, 5 Guardians, and 6 Evolvers per iteration. Conversely, our dynamic approach allocates more Explorers early to discover diverse reward structures and shifts toward Guardians and Evolvers later for precise optimization. Table 2 demonstrates that dynamic scheduling consistently improves the task-specific performance metrics on the IsaacGym tasks and the success rate on *Pen*. This confirms that transitioning from broad exploration to localized refinement enables fine-grained optimization of promising candidates and ensures robust final performance.

Table 2: Impact of Scheduling Strategy on task performance.

Scheduling Strategy	**Ant**	**FrankaCabinet**	**AllegroHand**	**Pen**
Fixed (5:5:6 Allocation)	6.5833 ± 0.3008	0.8157 ± 0.0206	20.7247 ± 1.9726	0.7277 ± 0.1685
Dynamic (Ours)	$\mathbf{7.2639 \pm 0.4584}$	$\mathbf{1.0000 \pm 0.0000}$	$\mathbf{23.4458 \pm 0.7224}$	$\mathbf{0.8967 \pm 0.0891}$

Impact of Global Elite Pool Size (K). We further investigate the sensitivity of R^2E to the capacity of the Global Elite Pool by varying the size $K \in \{3, 5, 8, 12, 16\}$ across four representative tasks. As shown in Table 3, setting $K = 5$ consistently yields the optimal performance. A smaller pool size restricts the structural diversity of the retained reward functions, often causing the search process to prematurely converge to local optima. Conversely, employing a larger pool under a fixed

generation budget lowers the admission threshold of the elite pool; it forces the framework to more frequently sample sub-optimal parents for crossover and mutation, which ultimately degrades both search efficiency and final performance.

Table 3: Ablation study on the size of the Global Elite Pool (K).

Task	$K = 3$	$K = 5$ **(Ours)**	$K = 8$	$K = 12$	$K = 16$
Ant	6.8632 ± 0.2850	$\mathbf{7.2639 \pm 0.4584}$	6.7844 ± 0.8383	5.5527 ± 0.4833	3.8366 ± 3.0366
FrankaCabinet	0.8621 ± 0.1026	$\mathbf{1.0000 \pm 0.0000}$	0.7425 ± 0.0156	0.7627 ± 0.1036	0.5163 ± 0.1673
AllegroHand	23.2724 ± 1.6273	$\mathbf{23.4458 \pm 0.7224}$	22.7741 ± 1.7614	21.6571 ± 1.6136	18.1642 ± 2.6115
Pen	0.8201 ± 0.0617	$\mathbf{0.8967 \pm 0.0891}$	0.8013 ± 0.0968	0.7314 ± 0.0916	0.5614 ± 0.2901

5.5 Qualitative Analysis & Case Studies

To provide a deeper understanding of the internal mechanisms of R^2E, we conduct a comprehensive qualitative analysis to answer three key research questions: **Q1.** Do the Explorer, Guardian, and Evolver roles exhibit distinct statistical characteristics consistent with their semantic definitions? **Q2.** How do these diverse roles collaborate to drive the evolutionary trajectory of the maximum reward? **Q3.** How does performance scale with the search horizon and the number of candidates in each iteration?

Analysis of Role Distinctiveness (Q1). Taking the *AllegroHand* task as a representative example, we analyze the statistical behaviors of the three roles to verify if they align with their semantic designs. As illustrated in Figure 6 (Left), the Explorer exhibits a lower mean reward ($\mu = 4.9$) but the highest variance, reflecting its aggressive search for novel reward structures that prioritizes diversity over immediate stability. In sharp contrast, the Guardian achieves the highest mean reward ($\mu = 18.8$) with minimal variance, confirming its role in performing conservative refinements to ensure reliable and consistent policy training. The Evolver demonstrates an intermediate statistical profile because it inherits high-performing structures from elite parents while introducing controlled variations via crossover and mutation, resulting in stability and average performance ($\mu = 17.7$) that fall between those of the Explorer and the Guardian. These distinct statistical signatures validate that the LLM effectively understands its assigned roles, demonstrating that our framework successfully transforms undifferentiated sampling into a structured search process. By ensuring that each specialized role performs its unique and complementary function, R^2E establishes a more robust paradigm that effectively expands the reward function search space.

Synergy and Evolutionary Trajectory (Q2). To better illustrate the coordination mechanism among roles during the evolution, we select a highly representative case study—the *LiftUnderarm* task, as visualized in Figure 6 (Right). In this task, the initial exploration phase without role assignment yields zero performance, highlighting the formidable challenges of reward design in the early stages. Nevertheless, these initial attempts are preserved in the Global Elite Pool. By analyzing these failed reward structures, the Explorer is able to identify ineffective patterns and innovatively design functional reward structures, thereby effectively expanding the search space. Subsequently, the Guardian refines and stabilizes the reward logic, leading to a significant performance leap. Finally, the Evolver leverages the strengths of elite candidates from the global history pool, combining the Explorer's exploration capability with the Guardian's stability to derive the globally optimal reward function. This evolutionary trajectory clearly demonstrates how the distinct roles collaborate to guide the search process from broad exploration to precise exploitation. Detailed results in Appendix B.

Search Horizon and Sampling Budget (Q3). We next examine whether the reported gains depend on a particular search budget. All other settings follow the main experiments. On *Ant*, *AllegroHand*, and *Pen*, we first fix the number of candidates in each iteration at $N = 16$ and vary only the number of iterations T. *Ant* and *AllegroHand* are measured by task-specific performance metrics, and *Pen* by success rate. As shown in Table 4, performance improves on every task as T increases from 1 to 7. Most of the gain appears by $T = 5$; extending the search to $T = 7$ brings a smaller additional improvement. We then fix $T = 5$ and vary only N. From the second iteration, the same dynamic schedule reallocates this budget in the order (Explorer, Guardian, Evolver). The four subsequent iterations use $(3, 0, 1) \to (2, 1, 1) \to (1, 1, 2) \to (0, 2, 2)$ when $N = 4$, $(6, 1, 1) \to (4, 2, 2) \to (2, 3, 3) \to (0, 4, 4)$ when $N = 8$, $(12, 2, 2) \to (8, 4, 4) \to (4, 6, 6) \to (0, 8, 8)$ when $N = 16$, and $(18, 3, 3) \to (12, 6, 6) \to (6, 9, 9) \to (0, 12, 12)$ when $N = 24$. Table 5 shows that sampling more candidates in each iteration also improves all three tasks. Increasing either the horizon or the

Table 4: Performance under different search horizons with $N = 16$. **Bold** indicates the highest value in each task. $T = 5$ is the setting used in the main experiments.

Iterations (T)	**Total candidates** ($T \times N$)	**Ant**	**AllegroHand**	**Pen**
1	16	4.8317 ± 0.3143	16.5829 ± 1.6192	0.3873 ± 0.1354
3	48	5.7638 ± 0.1865	20.6347 ± 1.0519	0.6516 ± 0.1017
5	80	7.2639 ± 0.4584	23.4458 ± 0.7224	0.8967 ± 0.0891
7	112	$\mathbf{7.6821 \pm 0.6817}$	$\mathbf{24.0685 \pm 0.4739}$	$\mathbf{0.9166 \pm 0.0539}$

Table 5: Performance under different numbers of candidates in each iteration with $T = 5$. **Bold** indicates the highest value in each task. $N = 16$ is the setting used in the main experiments.

Budget (N)	**Total candidates** ($5 \times N$)	**Ant**	**AllegroHand**	**Pen**
4	20	6.0789 ± 0.6078	19.0723 ± 0.4367	0.6049 ± 0.1033
8	40	6.5672 ± 1.1067	20.9287 ± 0.8713	0.7549 ± 0.0376
16	80	7.2639 ± 0.4584	23.4458 ± 0.7224	0.8967 ± 0.0891
24	120	$\mathbf{7.9834 \pm 0.2569}$	$\mathbf{24.4028 \pm 0.3644}$	$\mathbf{0.9534 \pm 0.0135}$

number of candidates therefore strengthens the search of R^{2}E. We set $N = 16$ and $T = 5$ in the main comparison so that the evaluation budget matches that of the baselines. Adapting T and N during search, rather than specifying them in advance, remains future work.

6 Conclusion

We propose R^{2}E, a role-driven reward evolutionary framework that transforms LLM-based reward design from undifferentiated sampling into a structured, collaborative search process. By orchestrating the Explorer, Guardian, and Evolver roles alongside a Global Elite Pool, R^{2}E explicitly balances broad exploration of the reward function space with the conservative refinement required for training stability. This architecture effectively mitigates optimization instability by preserving globally best-performing candidates as persistent references, maintaining a stable evolutionary trajectory across iterations. Experiments on IsaacGym and Bi-DexHands demonstrate that R^{2}E consistently outperforms baselines, especially in complex manipulation tasks. More importantly, R^{2}E pioneers a more effective paradigm for autonomous reward design. By assigning distinct role characteristics to the LLM, our framework structurally improves reward functions during the evolutionary process, enabling the search to effectively escape local optima and discover reward functions of significantly higher quality.

Beyond the present framework, an important question is how to retain the experience contained in failed reward functions and use it together with the experience of successful ones. This memory should persist across iterations, so that later rounds can avoid previously ineffective structures while continuing to build on rewards that have already proved useful. A related question is how to decide when evolution should stop. Both the joint use of failure and success across iterations and a principled stopping decision merit further attention in automated reward design.

Acknowledgments and Disclosure of Funding

This work was supported by the National Key Research and Development Program of China (No. 2025YB3003500) and the National Natural Science Foundation of China (No. U25A20437, 62672521, 62372473).

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[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] 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.

[4] Riad Akrour, Marc Schoenauer, and Michèle Sebag. April: Active preference learning-based reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 116–131. Springer, 2012.

[5] Andrew G Barto. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pages 17–47. Springer, 2012.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[7] Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Yuanpei Chen, Yiran Geng, Fangwei Zhong, Jiaming Ji, Jiechuang Jiang, Zongqing Lu, Hao Dong, and Yaodong Yang. Bi-dexhands: Towards human-level bimanual dexterous manipulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2804–2818, 2023.

[9] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[10] N Gao, X Zhang, X Jiang, et al. Rf-agent: Automated reward function design via language agent tree search. *arXiv preprint arXiv:2602.23876*, 2026.

[11] Alexander David Goldie, Zilin Wang, Jaron Cohen, Jakob Nicolaus Foerster, and Shimon Whiteson. How should we meta-learn reinforcement learning algorithms? *arXiv preprint arXiv:2507.17668*, 2025.

[12] RISHI HAZRA, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. REvolve: Reward evolution with large language models using human feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

[13] Changxin Huang, Yanbin Chang, Junfan Lin, Junyang Liang, Runhao Zeng, and Jianqiang Li. Efficient language-instructed skill acquisition via reward-policy co-evolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 14576–14584, 2025.

[14] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[15] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[16] Pengyi Li, Jianye HAO, Hongyao Tang, Yifu Yuan, Jinbin Qiao, Zibin Dong, and YAN ZHENG. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[17] 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. In *The Twelfth International Conference on Learning Representations*, 2024.

[18] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. In *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024.

[19] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. *GitHub repository, May*, 2021.

[20] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[21] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 663–670, 2000.

[22] 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. Citeseer, 1999.

[23] Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pages 77–95. Springer, 2020.

[24] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[27] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18 (136):1–46, 2017.

[28] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *ICLR*, 2024.

[29] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229, pages 374–404, 2023.

[30] Yuwei Zeng, Yao Mu, and Lin Shao. Learning reward for robot skills using large language models via self-alignment. In *International Conference on Machine Learning*, pages 58366–58386. PMLR, 2024.

[31] 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.

A Roles Engineering & Templates

In this section, we provide the detailed prompt templates used for R^2E. We leverage a chat-based interaction format consisting of **System**, **User**, and **Assistant** messages to guide the LLM.

A.1 Common Interface

To maintain a controlled experimental environment and ensure an identical informational baseline, the **User** message is kept strictly invariant across all assigned roles and throughout every iteration of the framework. This design choice ensures that the core task definition and environmental context delivered to the LLM remain homogeneous, regardless of the specific role-playing constraints or the current stage of the process. By employing a standardized template, we guarantee that the LLM parses the observation space and task goals from a constant source of truth, thereby isolating role-specific behavioral variances from the fundamental understanding of the task objectives.

Common User Template

```
The Python environment is {task_obs_code_string}.  Write a reward function for the
following task:  {task_description}.
```

A.2 Initialization

During the initial iteration, R^2E operates in an unconstrained manner without role-specific priors to facilitate a comprehensive exploration of the reward space. By prompting the LLM as a generalist reward engineer, we prioritize broad coverage of the potential reward space over specific behavioral alignment. This unconstrained phase is critical for generating a diverse set of reward candidates, which serves to seed the Global Elite Pool with a sufficiently varied population for subsequent evolutionary refinement.

Initial System Template

```
You are a reward engineer trying to write reward functions to solve reinforcement
learning tasks as effectively as possible.
Your goal is to write a free exploration reward function for the environment that will
help the agent learn the task described in text.
Your reward function should use useful variables from the environment as inputs.  As an
example, the reward function signature can be:  {task_reward_signature_string}
Since the reward function will be decorated with @torch.jit.script, please make sure
that the code is compatible with TorchScript (e.g., use torch tensor instead of numpy
array).
Make sure any new tensor or variable you introduce is on the same device as the input
tensors.

The output of the reward function should consist of two items:
   (1) the total reward,
   (2) a dictionary of each individual reward component.
The code output should be formatted as a Python code string:  "```python ...```".

Some helpful tips for writing the reward function code:
   (1) You may find it helpful to normalize the reward to a fixed range by applying
transformations like torch.exp to the overall reward or its components
   (2) If you choose to transform a reward component, then you must also introduce
a temperature parameter inside the transformation function; this parameter must be
a named variable in the reward function and it must not be an input variable.  Each
transformed reward component should have its own temperature variable
   (3) Make sure the type of each input variable is correctly specified; a float input
variable should not be specified as torch.Tensor
   (4) Most importantly, the reward code´s input variables must contain only attributes
of the provided environment class definition (namely, variables that have prefix self).
Under no circumstance can you introduce new input variables.
```

A.3 Role-Driven Evolution

From the second iteration, R^2E transitions to a structured evolutionary phase featuring three roles: Explorer, Guardian, and Evolver. All roles share an identical User Message to ensure task consistency. To optimize efficiency, their prompting structures are differentiated: the Explorer and Guardian utilize both System and Assistant messages (with elite rewards as exemplars), while the Evolver is streamlined to a System Message alone, which is sufficient to execute its mutation and crossover logic.

A.3.1 Explorer Role

The Explorer is designed to circumvent local optima by synthesizing novel reward structures. By utilizing elite rewards as negative references, the role is explicitly incentivized to pursue divergent candidates, thereby expanding the search breadth and maintaining the structural diversity of the reward space.

Explorer System Template

```
You are an EXPLORER reward engineer whose mission is to design innovative and effective
reward functions for reinforcement learning tasks.

Your goal is to create a TorchScript-compatible reward function that helps the agent
learn the task described in text, using meaningful environment variables as inputs.

As the EXPLORER, you should prioritize creativity, diversity, and unconventional ideas,
proposing exploratory or high-variance reward terms that may lead the agent to discover
novel behaviors or strategies.  However, to ensure the reward remains trainable,
all terms must respect the following safety constraints:  they must be numerically
reasonable (preferably within [-5.0, 5.0] per term), smooth and differentiable,
physically plausible, free of discontinuous or reward-hacking behaviors, and composed
only of TorchScript-safe tensor operations.  Innovation should come from structure—not
from unbounded magnitudes or unstable expressions.

As an example, the reward function signature can be:  {task_reward_signature_string}
```

Explorer Assistant Template

```
We have analyzed the current Global Elite Pool.  Below are two HIGH-PERFORMING but
CONVENTIONAL reward functions found so far.

[REFERENCE CODE A - Current Best]
{first_function}
{the_first_function_metrics}

[REFERENCE CODE B - Alternative High-Performer]
{second_function}
{the_second_function_metrics}

You are the EXPLORER for an RL task in robot problems.  You have been given task
descriptions, reference rewards, and policy feedback.

Your job:
 (1) Silently analyze the task and identify key success/failure behaviors.
 (2) Silently analyze the reference rewards and metrics:
      - Which behaviors they encourage or ignore.
      - Which failure modes (e.g., low success, high variance, instability) remain.
 (3) Then design a new reward that addresses these failure modes with better behavioral
     logic.

Hard constraints (YOU MUST FOLLOW):
 (1) Only use simple, stable ops:  distances, norms, squared terms, torch.clamp,
     torch.tanh, small torch.exp.
 (2) Do not manually expand quaternion algebra (Most Importantly!).
 (3) Do not create variables like w1, x1, y1, z1, rel_w, rel_x.
 (4) Do not use any variables that do not appear in the environment information.
```

```
Behavior & innovation requirements:
 (1) Change at least one key behavioral aspect vs the reference (e.g., new shaping
     term, safety margin, smoothness/efficiency bonus, curriculum-style reward).
 (2) Each reward component must correspond to a clear behavior (e.g., "approach goal",
     "stabilize orientation", "avoid collisions", "move smoothly").
 (3) Prefer logical changes in the reward (adding/removing components or changing
     activation conditions) over only re-scaling existing weights.

Output:  Only the final PyTorch reward code, no explanations.
```

A.3.2 Guardian Role

The Guardian focuses on enhancing training stability and ensuring performance robustness through conservative refinements. It utilizes the current best-performing reward function as a positive reference to optimize numerical scales and smoothness while strictly maintaining the core behavioral logic.

Guardian System Template

```
You are a GUARDIAN reward engineer whose mission is to design robust, stable, and
generalizable reward functions for reinforcement learning tasks.

Your goal is to create a TorchScript-compatible reward function that helps the agent
learn the task described in text, using meaningful environment variables as inputs.

As the GUARDIAN, you should focus on well-established, reliable reward design
principles that support consistent learning across environments, avoid risky or overly
complex structures, ensure numerical stability, and prevent reward-hacking behavior.
Prioritize clarity, simplicity, and physically grounded reward terms that promote
steady and predictable policy improvement.

As an example, the reward function signature can be:  {task_reward_signature_string}
```

Guardian Assistant Template

```
The following is the CURRENT BEST (Top-1) reward function found in the Global Elite
Pool.

[TARGET CODE - Do Not Change Structure]
{best_function}
{the_best_function_metrics}

You are the GUARDIAN for an RL task in robot problems.

You have been given:
 (1) The current BEST (Top-1) reward function from the Global Elite Pool.
 (2) Task description and observations.
 (3) Policy feedback and metrics for this reward (e.g., success rate, return curves,
     variance, qualitative behaviors).

Your job:
 (1) Silently analyze how this reward shapes behavior and which parts are already
     working well.
 (2) Silently analyze the metrics and behaviors:
       - Signs of instability:  high variance, jittery actions, oscillations, frequent
         failures.
       - Signs of imbalance:  one component dominating, others nearly constant or
         ineffective.
 (3) Then refine this reward to improve stability and robustness without breaking its
     core task logic.

Hard constraints (YOU MUST FOLLOW):
 (1) Only use simple, stable ops:  distances, norms, squared terms, torch.clamp,
     torch.tanh, small torch.exp.
```

```
  (2) Do not manually expand quaternion algebra (Most Importantly!).
  (3) Do not create variables like w1, x1, y1, z1, rel_w, rel_x, rel_y, rel_z (Most
      Importantly!).

Behavior & refinement requirements:
  (1) Keep the main task objective of the Top-1 reward; the structure may be modified,
      but its logic must remain unchanged.
  (2) Focus on:
        - Stability and smoothness (e.g., small penalties on velocities / action changes,
          clamping extreme rewards).
        - Magnitude balancing (rescale coefficients so components have reasonable
          relative impact).
        - Making ineffective/static components sensitive to state changes (adjust
          scales/thresholds instead of deleting them).
  (3) You may add small regularization/safety terms, clamp existing terms, and retune
      weights, but avoid introducing completely new, unrelated sub-goals.

Output:  Only the final PyTorch reward code, no explanations.
```

A.3.3 Evolver Role

The Evolver facilitates structured inheritance and continuous optimization by treating high-performing reward functions as genotypes for recombination. It merges complementary components from multiple elite rewards to synthesize hybrid offspring that inherit successful behavioral traits while applying controlled mutations to explore improved formulations.

Evolver System Template

```
You are an EVOLVER whose mission is to perform evolutionary optimization on reward
functions across training iterations.

After each evaluation cycle, you must select the two best-performing parent reward
functions from the archive and generate a new offspring reward function by applying
structured crossover and controlled mutation to their components.  Your design must
preserve the useful characteristics of the parents while introducing meaningful
variations that may improve future performance.  Ensure that all generated reward
components remain TorchScript-compatible, numerically stable, physically plausible, and
consistent with the environment's state variables.  Your goal is to balance inheritance
and innovation, producing offspring that are both grounded in successful prior designs
and capable of exploring new reward structures.

For clarity, the two parent functions you will operate on are provided below:

The first parent function:
{first_function}
{the_first_function_metrics}

The second parent function:
{second_function}
{the_second_function_metrics}


You have been given:
  (1) Two parent reward functions (Parent A and Parent B).
  (2) Their training metrics and qualitative behavior summaries.
  (3) The task description and observations.
  (4) The target function signature:  {task_reward_signature_string}.

Your job:
  (1) Silently compare the two parents:
        - Which terms help stability (low variance, smooth motion, fewer failures).
        - Which terms help task performance (success rate, high returns, fast
          completion).
        - Which terms are redundant, static, or harmful.
  (2) Design ONE offspring reward by:
```

```
      - Selecting useful components from Parent A and Parent B (crossover).
      - Applying 1-3 small but meaningful mutations to improve behavior (e.g., weight
        tuning, smoothness/safety penalties, thresholds/clamps, shaping adjustments).

Behavior & evolution requirements:
 (1) Preserve the main task objective and core logic shared by the parents.
 (2) Inherit at least one strong/working behavior component from EACH parent, unless it
     is clearly harmful.
 (3) Make the offspring more stable and/or more effective than both parents by:
      - Balancing magnitudes of components,
      - Adding or strengthening smoothness/safety/shaping terms,
      - Slightly adjusting weights or nonlinearities.
 (4) Every reward component must correspond to a clear behavior (e.g., "approach goal",
     "stabilize orientation", "avoid collisions", "move smoothly").

Output:  Only the final PyTorch reward code, no explanations.
```

B R^2E Case Example

We take *LiftUnderarm* as an example to analyze in detail how the roles in R^2E perform reward function design.

In the initial iteration, R^2E operates without any role constraints and generates an initial reward function.

Initial Reward Function Code

```python
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2, dim=-1)

    # Encourage both hands to be close to their respective handles
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    hand_handle_temp = 1.0
    reward_hand_handle = torch.exp(-hand_handle_dist / hand_handle_temp)

    # Distance from object to goal
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2, dim=-1)
    obj_goal_temp = 0.1
    reward_obj_goal = torch.exp(-obj_to_goal_dist / obj_goal_temp)

    # Penalize large object linear and angular velocities (to encourage stability at goal)
    lin_vel_penalty = torch.norm(object_linvel, p=2, dim=-1)
    ang_vel_penalty = torch.norm(object_angvel, p=2, dim=-1)
    vel_penalty = lin_vel_penalty + ang_vel_penalty
    vel_temp = 0.5
    reward_stability = torch.exp(-vel_penalty / vel_temp)

    # Total reward as a weighted combination
    total_reward = ( 1.0 * reward_hand_handle + 2.0 * reward_obj_goal + 0.5 * reward_stability )
```

```
    reward_components = {
        "reward_hand_handle": reward_hand_handle,
        "reward_obj_goal": reward_obj_goal,
        "reward_stability": reward_stability
    }

    return total_reward, reward_components
```

In the second and third iterations of reward function generation, the Explorer deliberately deviates from the current optimal reward structure based on existing reward function paradigms. The final generated reward function is presented as follows.

Explorer

```
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)

    # Grasp reward: strong incentive to get both hands close to handles
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    reward_grasp = torch.exp(-hand_handle_dist * 8.0)

    # Object-to-goal distance with progressive shaping: always active but stronger when closer
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2.0, dim=-1)
    reward_lift = torch.exp(-obj_to_goal_dist * 5.0)

    # Early-stage bonus for upward movement when object is still low
    object_height = object_pos[:, 2]  # z-coordinate
    goal_height = goal_pos[:, 2]
    below_goal = (object_height < goal_height - 0.1).float()
    upward_vel = torch.clamp(object_linvel[:, 2], min=0.0)
    reward_upward = below_goal * upward_vel * 2.0

    # Stability bonus only when near the goal (encourage settling)
    near_goal = (obj_to_goal_dist < 0.2).float()
    lin_vel_norm = torch.norm(object_linvel, p=2.0, dim=-1)
    ang_vel_norm = torch.norm(object_angvel, p=2.0, dim=-1)
    total_vel = lin_vel_norm + ang_vel_norm
    stability_bonus = near_goal * torch.exp(-total_vel * 4.0)

    # Symmetry term to prevent one-handed grasping
    hand_diff = torch.abs(left_hand_to_left_handle - right_hand_to_right_handle)
    symmetry_bonus = torch.exp(-hand_diff * 6.0)

# Total reward composition
    total_reward = ( 1.0 * reward_grasp + 2.0 * reward_lift + 1.0 * reward_upward +
                     1.5 * stability_bonus + 0.5 * symmetry_bonus )

    reward_components = {
        "reward_grasp": reward_grasp,
```

```
        "reward_lift": reward_lift,
        "reward_upward": reward_upward,
        "stability_bonus": stability_bonus,
        "symmetry_bonus": symmetry_bonus
    }

    return total_reward, reward_components
```

The Guardian is designed to fine-tune the current best reward functions. As can be observed from the reward function generated by the Explorer, the Guardian adjusts three key components in the fourth iteration: it reduces the exponential decay rate to slow down attenuation and expand the guidance range, sets stricter boundaries and thresholds to prevent reward exploitation and early triggering, and rebalances the weight distribution of the overall reward.

Guardian

```
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)

    # Grasp reward: strong incentive to get both hands close to handles
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    reward_grasp = torch.exp(-hand_handle_dist * 6.0)

    # Object-to-goal distance with progressive shaping: always active but stronger when closer
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2.0, dim=-1)
    reward_lift = torch.exp(-obj_to_goal_dist * 4.0)

    # Early-stage bonus for upward movement when object is still low
    object_height = object_pos[:, 2]  # z-coordinate
    goal_height = goal_pos[:, 2]
    below_goal = (object_height < goal_height - 0.1).float()
    upward_vel = torch.clamp(object_linvel[:, 2], min=0.0, max=1.0)
    reward_upward = below_goal * upward_vel * 1.5

    # Stability bonus only when near the goal (encourage settling)
    near_goal = (obj_to_goal_dist < 0.25).float()
    lin_vel_norm = torch.norm(object_linvel, p=2.0, dim=-1)
    ang_vel_norm = torch.norm(object_angvel, p=2.0, dim=-1)
    total_vel = lin_vel_norm + ang_vel_norm
    stability_bonus = near_goal * torch.exp(-total_vel * 3.0)

    # Symmetry term to prevent one-handed grasping
    hand_diff = torch.abs(left_hand_to_left_handle - right_hand_to_right_handle)
    symmetry_bonus = torch.exp(-hand_diff * 5.0)

    # Total reward composition with balanced weights
    total_reward = ( 1.2 * reward_grasp + 2.0 * reward_lift + 0.8 * reward_upward +
                     1.2 * stability_bonus + 0.6 * symmetry_bonus )
```

```
    reward_components = {
        "reward_grasp": reward_grasp,
        "reward_lift": reward_lift,
        "reward_upward": reward_upward,
        "stability_bonus": stability_bonus,
        "symmetry_bonus": symmetry_bonus
    }

    return total_reward, reward_components
```

Evolver generates new reward functions based on component-level crossover and mutation mechanisms. Specifically, it selects the current optimal reward function from the Global Elite Pool and iteratively combines it with another randomly chosen reward function from the pool. In this generation process, the reference reward functions utilized by Evolver are as follows.

The first reward function referenced by the Evolver

```
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)

    # Grasp reward: strong incentive to get both hands close to handles
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    reward_grasp = torch.exp(-hand_handle_dist * 6.0)

    # Object-to-goal distance with progressive shaping: always active but stronger when closer
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2.0, dim=-1)
    reward_lift = torch.exp(-obj_to_goal_dist * 4.0)

    # Early-stage bonus for upward movement when object is still low
    object_height = object_pos[:, 2]  # z-coordinate
    goal_height = goal_pos[:, 2]
    below_goal = (object_height < goal_height - 0.1).float()
    upward_vel = torch.clamp(object_linvel[:, 2], min=0.0, max=1.0)
    reward_upward = below_goal * upward_vel * 1.5

    # Stability bonus only when near the goal (encourage settling)
    near_goal = (obj_to_goal_dist < 0.25).float()
    lin_vel_norm = torch.norm(object_linvel, p=2.0, dim=-1)
    ang_vel_norm = torch.norm(object_angvel, p=2.0, dim=-1)
    total_vel = lin_vel_norm + ang_vel_norm
    stability_bonus = near_goal * torch.exp(-total_vel * 3.0)

    # Symmetry term to prevent one-handed grasping
    hand_diff = torch.abs(left_hand_to_left_handle - right_hand_to_right_handle)
    symmetry_bonus = torch.exp(-hand_diff * 5.0)

    # Total reward composition with balanced weights
    total_reward = ( 1.2 * reward_grasp + 2.0 * reward_lift + 0.8 * reward_upward +
```

```
                    1.2 * stability_bonus + 0.6 * symmetry_bonus )

    reward_components = {
        "reward_grasp": reward_grasp,
        "reward_lift": reward_lift,
        "reward_upward": reward_upward,
        "stability_bonus": stability_bonus,
        "symmetry_bonus": symmetry_bonus
    }

    return total_reward, reward_components
```

The second reward function referenced by the Evolver

```
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)

    # Grasp reward with smooth ramp-up instead of pure exponential
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    reward_grasp = 1.0 - torch.tanh(hand_handle_dist * 5.0)

    # Lifting progress measured as vertical distance above initial height (assumed 0.5)
    object_height = object_pos[:, 2]
    lift_progress = torch.clamp(object_height - 0.5, min=0.0, max=0.4)
    reward_lift = lift_progress * 4.0

    # Goal proximity with soft activation based on both height and distance
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2.0, dim=-1)
    height_sufficient = torch.sigmoid((object_height - 0.6) * 20.0)  # smooth gate
    reward_goal = height_sufficient * torch.exp(-obj_to_goal_dist * 8.0)

    # Stability bonus only when near goal and lifted high enough
    near_goal = (obj_to_goal_dist < 0.25).float()
    sufficiently_lifted = (lift_progress > 0.2).float()
    lin_vel_norm = torch.norm(object_linvel, p=2.0, dim=-1)
    ang_vel_norm = torch.norm(object_angvel, p=2.0, dim=-1)
    total_vel = lin_vel_norm + ang_vel_norm
    stability_bonus = near_goal * sufficiently_lifted * torch.exp(-total_vel * 5.0)

    # Symmetry term with stronger penalty for large imbalance
    hand_diff = torch.abs(left_hand_to_left_handle - right_hand_to_right_handle)
    symmetry_bonus = torch.exp(-hand_diff * 8.0)

    # Smoothness bonus: penalize excessive hand acceleration (approx via velocity magnitude)
    # Approximate hand velocities by finite difference is not available; instead use fingertip vel
    #proxy indirectly via object motion coupling
    # Instead, add efficiency term: discourage excessive hand movement once grasped
    grasp_achieved = (hand_handle_dist < 0.2).float()
```

```
    hand_movement = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1) +
                    torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)
    efficiency_bonus = grasp_achieved * torch.exp(-hand_movement * 3.0)

    # Total reward composition
    total_reward = ( 1.2 * reward_grasp + 1.5 * reward_lift + 2.0 * reward_goal +
                     1.8 * stability_bonus + 0.6 * symmetry_bonus + 0.5 * efficiency_bonus )

    reward_components = {
        "reward_grasp": reward_grasp,
        "reward_lift": reward_lift,
        "reward_goal": reward_goal,
        "stability_bonus": stability_bonus,
        "symmetry_bonus": symmetry_bonus,
        "efficiency_bonus": efficiency_bonus
    }

    return total_reward, reward_components
```

Specifically, the Evolver extracts functional components from the first reference reward function, then fuses and recombines them with functional components from the second reference reward function via crossover and mutation mechanisms to construct the final reward function, as shown below.

Evolver

```
def compute_reward(
    object_pos: torch.Tensor,
    goal_pos: torch.Tensor,
    left_hand_pos: torch.Tensor,
    right_hand_pos: torch.Tensor,
    pot_left_handle_pos: torch.Tensor,
    pot_right_handle_pos: torch.Tensor,
    object_linvel: torch.Tensor,
    object_angvel: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Distance from hands to respective handles
    left_hand_to_left_handle = torch.norm(left_hand_pos - pot_left_handle_pos, p=2.0, dim=-1)
    right_hand_to_right_handle = torch.norm(right_hand_pos - pot_right_handle_pos, p=2.0, dim=-1)

    # Grasp reward: hybrid of exponential and tanh for smoothness and strong early signal
    hand_handle_dist = left_hand_to_left_handle + right_hand_to_right_handle
    reward_grasp = torch.exp(-hand_handle_dist * 5.0) * (1.0 - torch.tanh(hand_handle_dist * 3.0))

    # Object-to-goal distance with height-aware activation
    obj_to_goal_dist = torch.norm(object_pos - goal_pos, p=2.0, dim=-1)
    object_height = object_pos[:, 2]
    goal_height = goal_pos[:, 2]
    height_sufficient = torch.sigmoid((object_height - (goal_height - 0.1)) * 15.0)
    reward_goal = height_sufficient * torch.exp(-obj_to_goal_dist * 6.0)

    # Upward progress bonus during initial lift phase
    initial_height = 0.5
    lift_progress = torch.clamp(object_height - initial_height, min=0.0, max=0.4)
    below_goal_phase = (object_height < goal_height - 0.05).float()
    upward_vel = torch.clamp(object_linvel[:, 2], min=0.0, max=1.0)
    reward_upward = below_goal_phase * upward_vel * 1.2

    # Stability bonus only when near goal and sufficiently lifted
    near_goal = (obj_to_goal_dist < 0.22).float()
    sufficiently_lifted = (lift_progress > 0.15).float()
```

```
    lin_vel_norm = torch.norm(object_linvel, p=2.0, dim=-1)
    ang_vel_norm = torch.norm(object_angvel, p=2.0, dim=-1)
    total_vel = lin_vel_norm + ang_vel_norm
    stability_bonus = near_goal * sufficiently_lifted * torch.exp(-total_vel * 4.0)

    # Symmetry bonus to encourage balanced two-handed grasp
    hand_diff = torch.abs(left_hand_to_left_handle - right_hand_to_right_handle)
    symmetry_bonus = torch.exp(-hand_diff * 6.0)

    # Efficiency bonus: discourage unnecessary hand movement after grasp
    grasp_achieved = (hand_handle_dist < 0.25).float()
    hand_movement = left_hand_to_left_handle + right_hand_to_right_handle
    efficiency_bonus = grasp_achieved * torch.exp(-hand_movement * 2.5)

    # Total reward composition with tuned weights
    total_reward = ( 1.3 * reward_grasp + 1.8 * reward_goal + 0.7 * reward_upward +
                     1.4 * stability_bonus + 0.5 * symmetry_bonus + 0.4 * efficiency_bonus )

    reward_components = {
        "reward_grasp": reward_grasp,
        "reward_goal": reward_goal,
        "reward_upward": reward_upward,
        "stability_bonus": stability_bonus,
        "symmetry_bonus": symmetry_bonus,
        "efficiency_bonus": efficiency_bonus
    }

    return total_reward, reward_components
```

Through the above evolutionary example of reward functions, we clearly demonstrate how the Explorer, Guardian, and Evolver collaborate to complete the optimization process. This process not only fully reflects the unique design and functional positioning of each role, but also further verifies the rationality and effectiveness of the method proposed in this paper.

C Environment Details

In this section, we provide detailed specifications for the experimental environments used in our evaluation. To ensure a comprehensive assessment across varying degrees of complexity, we utilize two distinct GPU-accelerated benchmarks: *NVIDIA IsaacGym* for standard locomotion and single-arm manipulation, and *Bi-DexHands* for high-dimensional bimanual coordination. The following subsections outline the specific observation spaces, action dimensions, and objectives for the tasks selected from each suite.

C.1 IsaacGym

IsaacGym [20] offers a high-performance learning platform designed to train policies for a wide variety of robotics tasks directly on the GPU. A key architectural advantage is that both the physics simulation and the neural network policy training reside on the GPU; the framework communicates by directly passing data from physics buffers to PyTorch tensors without ever traversing CPU bottlenecks. This end-to-end architecture leads to drastically accelerated training times—achieving 2–3 orders of magnitude improvements compared to conventional reinforcement learning setups that rely on CPU-based simulators. We leverage this high-throughput capability to evaluate R^2E across a diverse suite of benchmarks, ranging from agile locomotion to complex manipulation, which utilize the specific observation and action spaces defined in Table 6.

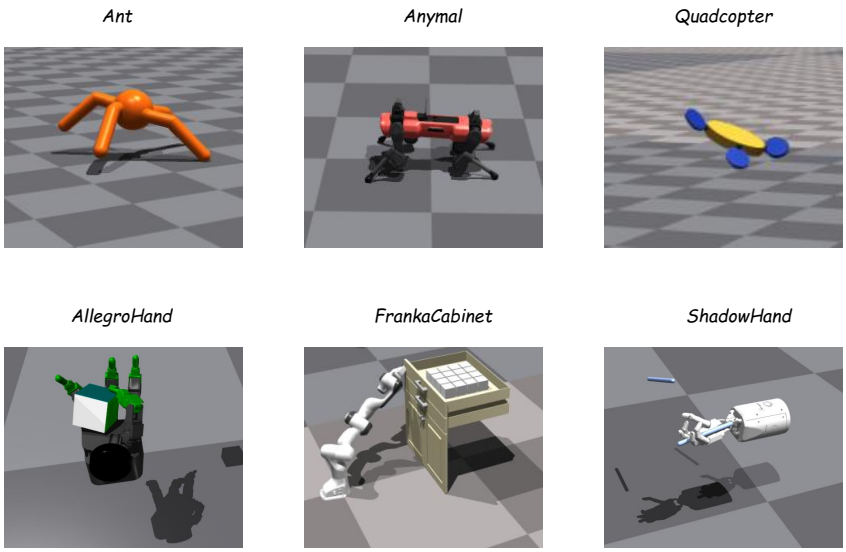


Figure 7: **Visualization of IsaacGym Tasks.** The benchmark suite is categorized into locomotion tasks (*Ant*, *Anymal*, *Quadcopter*) and manipulation tasks (*AllegroHand*, *FrankaCabinet*, *ShadowHand*).

Table 6: **Specifications of IsaacGym Environments.** Summary of observation (Obs.) and action (Act.) dimensions along with primary task objectives.

Task	Obs.	Act.	Task Description
Ant	60	8	Maximize forward velocity while maintaining stability.
Anymal	48	12	Track random linear and angular velocity commands.
Quadcopter	21	12	Navigate to target coordinates and maintain stable hover.
AllegroHand	88	16	Reorient object to target pose using a 16-DoF hand.
FrankaCabinet	23	9	Interact with and open a cabinet drawer.
ShadowHand	211	20	Reorient object using a 24-DoF anthropomorphic hand.

C.2 Bi-DexHands

Bi-DexHands [8] represents a significant escalation in control complexity, offering a comprehensive library of bimanual dexterous manipulation tasks built upon the high-performance IsaacGym simulator. Designed to match the dexterity of human bimanual skills, this benchmark necessitates the precise coordination of two multi-joint Shadow Hands to solve intricate cooperation challenges in a shared workspace. As illustrated in Figure 8, the selected environments span a wide spectrum of contact-rich scenarios—ranging from collaborative object handover to coordinated lifting and throwing—thereby serving as a rigorous testbed for evaluating policy synthesis in high-dimensional, multi-agent continuous control spaces.

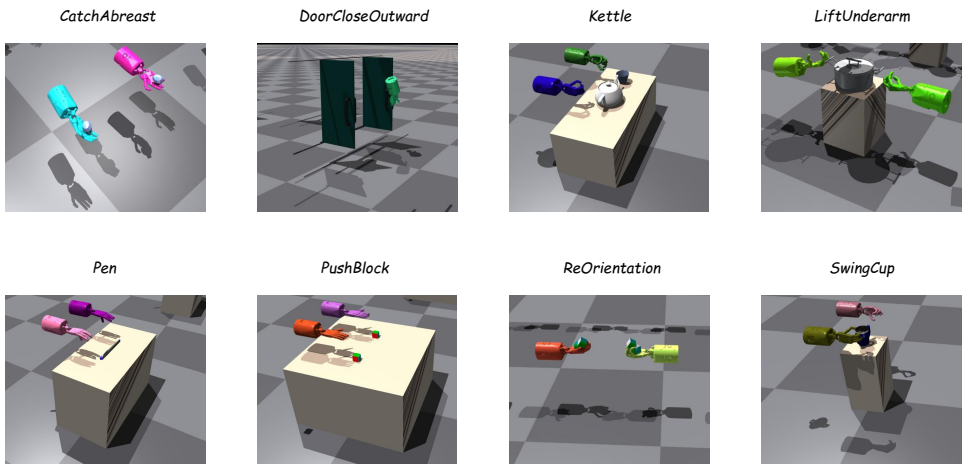


Figure 8: **Visualization of Bi-DexHands Tasks.** These tasks require synchronized control of two agents to solve high-dimensional manipulation problems, such as pouring water (*Kettle*) or collaborative swinging (*SwingCup*).

Table 7: **Detailed Specifications of Bi-DexHands Environments.** The table lists the high-dimensional observation (Obs.) and action (Act.) spaces alongside the specific coordination logic for each task.

Task	Obs.	Act.	Task Description
CatchAbreast	422	52	This class corresponds to the Catch Abreast task. This environment consists of two shadow hands placed side by side in the same direction and an object that needs to be passed. Compared with the previous environment which is more like passing objects between the hands of two people, this environment is designed to simulate the two hands of the same person passing objects, so different catch techniques are also required and require more hand translation and rotation techniques
DoorCloseOutward	417	52	This class corresponds to the DoorCloseOutward task. This environment also require a closed door to be opened and the door can only be pushed inward or initially open outward, but because they can't complete the task by simply pushing, which need to catch the handle by hand and then open or close it, so it is relatively difficult
Kettle	417	52	This class corresponds to the PourWater task. This environment involves two hands, a kettle, and a bucket, we need to hold the kettle with one hand and the bucket with the other hand, and pour the water from the kettle into the bucket. In the practice task in IsaacGym, we use many small balls to simulate the water
LiftUnderarm	417	52	This class corresponds to the LiftUnderarm task. This environment requires grasping the pot handle with two hands and lifting the pot to the designated position. This environment is designed to simulate the scene of lift in daily life and is a practical skill
Pen	417	52	This class corresponds to the Open Pen Cap task. This environment involves two hands and a pen, we need to use two hands to open the pen cap
PushBlock	428	52	This class corresponds to the PushBlock task. This environment involves two hands and two blocks, we need to use both hands to reach and push the block to the desired goal separately. This is a relatively simple task

Table 7 – continued from previous page

Task	Obs.	Act.	Task Description
ReOrientation	422	40	This class corresponds to the ReOrientation task. This environment involves two hands and two objects. Each hand holds an object and we need to reorient the object to the target orientation
SwingCup	417	52	This class corresponds to the SwingCup task. This environment involves two hands and a dual handle cup, we need to use two hands to hold and swing the cup together

D Implementation Details

This section provides the comprehensive experimental configurations required for reproducibility, covering the massively parallel simulation settings, LLM sampling strategies, and the robust code execution pipeline.

D.1 RL Training Configuration

We utilize the Proximal Policy Optimization (PPO) algorithm [19] as the core policy optimizer. To ensure rigorous reproducibility and isolate the contribution of the evolved reward functions, we strictly adhere to the default hyperparameter configurations (e.g., network architecture, learning rate, clipping range) provided in the official IsaacGym and Bi-DexHands repositories. Consequently, no task-specific tuning was performed on the PPO hyperparameters.

The primary computational variable adjusted across tasks is the number of parallel environments (N_{envs}). This parameter is optimized to fully saturate the GPU memory and maximize simulation throughput, thereby accelerating the fitness evaluation of reward candidates. The specific environment counts for all evaluated tasks are enumerated in Table 8.

Table 8: **Parallel Environment Configurations.** The specific number of parallel simulation instances (N_{envs}) utilized for each task to maximize GPU throughput.

IsaacGym Tasks		**Bi-DexHands Tasks**	
Task Name	N_{envs}	**Task Name**	N_{envs}
AllegroHand	16,384	CatchAbreast	2,048
Quadcopter	8,192	DoorCloseOutward	2,048
ShadowHand	8,192	Kettle	2,048
Ant	4,096	LiftUnderarm	2,048
Anymal	4,096	Pen	2,048
FrankaCabinet	4,096	PushBlock	2,048
		ReOrientation	2,048
		SwingCup	2,048

D.2 LLM Generation & Evolutionary Settings

To ensure full reproducibility and facilitate future research, we detail the stochastic sampling configurations and the evolutionary hyperparameters used to instantiate the R^2E framework.

Sampling Parameters We leverage the *Qwen3-Max* model via its high-throughput completion API. The sampling strategy is designed to balance the generation of syntactically correct code with the creativity required for reward discovery:

- **Temperature ($T_{llm} = 1.0$):** We maintain a constant temperature of 1.0 across all roles (*Explorer*, *Guardian*, *Evolver*). Unlike standard code generation tasks that prefer lower temperatures for precision, reward synthesis requires traversing a non-convex optimization landscape. This higher entropy setting prevents the model from converging prematurely to conventional coding patterns, ensuring sufficient diversity in the structural logic and coefficient selection.
- **Batch Sampling (n):** To minimize wall-clock latency, we utilize the *n* parameter to trigger parallel batch sampling. Each role's assigned budget is generated within a single request context, effectively

utilizing the provider's limits (600 RPM / 1M TPM). This approach reduces the overhead associated with establishing multiple HTTP connections.

- **Max Output Tokens (64k):** The output limit is set to 64,000 tokens. This generous upper bound is critical for the *Evolver* role, which often engages in "Chain-of-Thought" reasoning before outputting the final code. It ensures that complex reward functions, which may include extensive logic branches and internal state tracking, are never truncated mid-generation.
- **Context Window (256k):** We utilize the full 256k context window to implement a "long-term memory" mechanism. This allows the prompt to include a comprehensive history of the *Global Elite Pool*, detailed error logs from failed executions, and multiple high-performing reference functions from previous iterations without exceeding input constraints.

Evolutionary Search Configuration The evolutionary process is governed by a set of structural constants empirically selected to balance computational frugality with search depth:

- **Sample Budget ($N = 16$):** We generate 16 candidate reward functions per iteration. This number provides a statistically significant sample size to cover different mutation directions while keeping the simulation cost manageable.
- **Search Horizon ($T = 5$):** The optimization runs for 5 iterative rounds. Our preliminary experiments indicated that marginal gains diminish significantly after the 5th iteration, making this a cost-effective stopping criterion.
- **Global Elite Pool Size ($K = 5$):** The Top-5 performing rewards are retained in the Elite Pool. This strategy prevents catastrophic forgetting by ensuring that the best genes are always available for crossover in subsequent generations.

Dynamic Role Scheduling Strategy From the second iteration ($t > 1$) onwards, the total sample budget $N = 16$ is dynamically partitioned among the specialized roles. This partition follows a linear decay schedule designed to mimic a *Simulated Annealing* process. Let $t \in \{2, \dots, T\}$ be the current iteration index. The allocation is calculated as follows:

$$N_{\text{Explorer}}(t) = \left\lceil N \times \left(1 - \frac{t-1}{T-1}\right) \right\rceil \tag{4}$$

$$N_{\text{Guardian}}(t) = \left\lfloor \frac{N - N_{\text{Explorer}}(t)}{2} \right\rfloor \tag{5}$$

$$N_{\text{Evolver}}(t) = N - N_{\text{Explorer}}(t) - N_{\text{Guardian}}(t) \tag{6}$$

Theoretical Interpretation: This scheduling strategy facilitates a smooth transition from *Exploration* to *Exploitation*. In the early stages (e.g., $t = 2$), N_{Explorer} dominates, encouraging the framework to cast a wide net and discover novel reward structures. As t approaches T, the budget shifts towards the *Guardian* and *Evolver*, focusing resources on refining the numerical stability of existing elites and performing local crossover operations. At $t = 1$, to maximize initial coverage, all N samples are generated via a role-agnostic initialization prompt.

E Additional Experimental Results

E.1 Evolution of the Best-So-Far Performance

To provide a more granular analysis of the reward search efficiency, we track the best performance discovered up to iteration t, averaged over 5 random seeds, across 14 tasks from IsaacGym and Bi-DexHands. Unlike the learning curve of the reward evaluated at the current iteration, this quantity records the best reward found so far. On IsaacGym it is the task-specific performance metric; on Bi-DexHands it is the success rate. It serves as a direct proxy for optimization efficiency and for the ability to escape local optima.

The results for IsaacGym and Bi-DexHands are illustrated in Figure 9 and Figure 10, respectively.

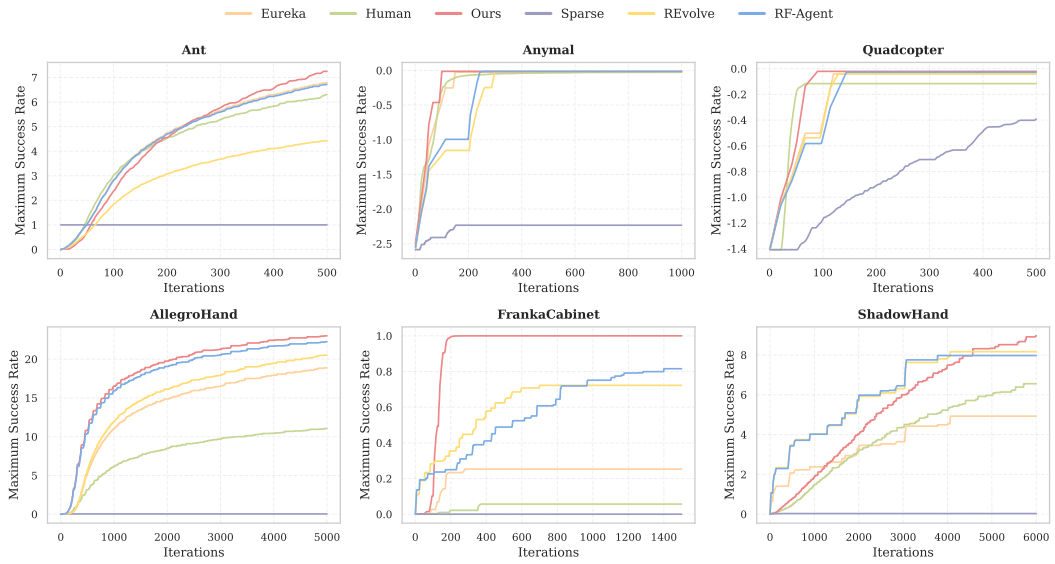


Figure 9: **Evolution of the best task-specific performance metric on IsaacGym.** The curves represent the highest task-specific performance metric achieved by the best reward discovered so far as the number of iterations increases. Our method (R^2E) demonstrates faster convergence (e.g., *FrankaCabinet*) and higher asymptotic performance (e.g., *AllegroHand*, *ShadowHand*) compared to baselines.

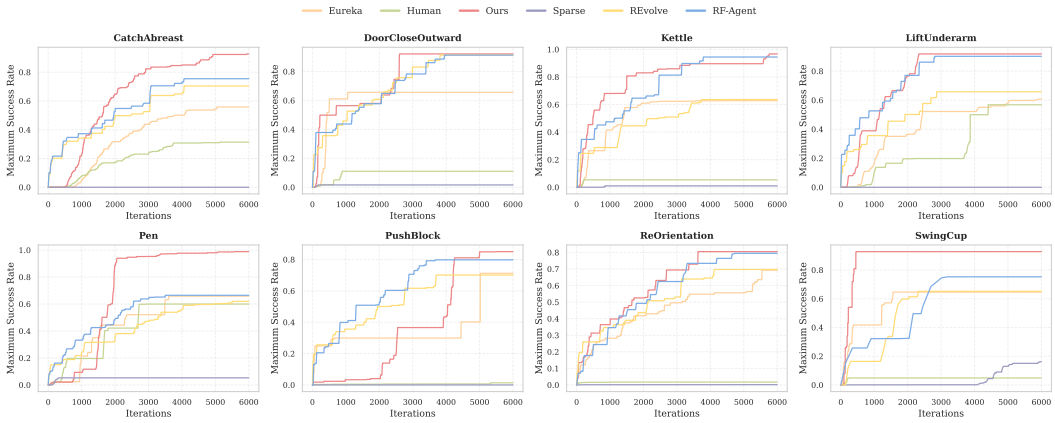


Figure 10: **Evolution of Maximum Success Rate on Bi-DexHands Tasks.** Comparisons across 8 complex manipulation tasks show that R^2E consistently locates superior reward functions earlier in the search process. Notably, in difficult tasks like *Pen*, R^2E succeeds in finding functional rewards where baselines stagnate.

Convergence Speed. As evident from the figures, R^2E exhibits significantly faster convergence rates compared to the baselines across the majority of tasks. In high-dimensional environments, our method rapidly identifies high-quality reward regions within the first few iterations. For instance, in *FrankaCabinet* (Fig. 9), R^2E achieves a steep performance jump almost immediately, whereas baselines requires significantly more iterations to reach comparable levels. Similarly, in *SwingCup* (Fig. 10), our method establishes a clear lead early in the search. This accelerated discovery is attributed to the Explorer role, which actively diversifies the search space, and the Global Elite Pool, which ensures that promising structures are immediately preserved and used to seed subsequent generations.

Asymptotic Performance and Local Optima Escaping. In addition to search efficiency, R^2E reaches a higher final task-specific performance metric on IsaacGym and a higher final success rate on Bi-DexHands. In tasks where baselines plateau early—indicating convergence to local optima—our approach continues to improve. This is most strikingly observed in *CatchAbreast* (Fig. 10), where baselines fail to make meaningful progress, while R^2E successfully discovers reward functions that solve the task. Furthermore, in fine-grained manipulation tasks such as *PushBlock* (Fig. 10) and *Kettle* (Fig. 10), R^2E widens the performance gap in later iterations. This sustained improvement demonstrates the effectiveness of the Guardian and Evolver roles, which continuously refine and recombine elite traits to push the performance boundary beyond what standard iterative prompting can achieve.

E.2 Computational Cost Analysis

The computational efficiency of R^2E is primarily determined by the parallelization capability of the RL training stage within IsaacGym. While the LLM generation phase (via *Qwen3-Max*) is relatively constant (taking approximately 1–3 minutes per iteration), the RL training time scales with the complexity of the environment's physics and the dimensionality of the action space.

Total Wall-clock Time Based on our experiments conducted on a 4× *NVIDIA RTX 4090* setup, the total time required to complete a full evolutionary search (5 iterations, $N = 16$ candidates per iteration) varies across tasks. We categorize the empirical execution times as follows:

- **Low Complexity (Locomotion/Control):** Tasks such as *Ant* and *Quadcopter* converge very rapidly, completing the entire 5-iteration search in approximately **45–50 minutes**.
- **Medium Complexity (Basic Manipulation):** Tasks involving robotic arms or basic legged locomotion, such as *FrankaCabinet* and *Anymal*, require approximately **1.5–2 hours**.
- **High Complexity (Dexterous Manipulation):** Intricate tasks using the Shadow Hand or Allegro Hand (e.g., *Pen*, *AllegroHand*) exhibit the highest computational demand due to complex contact physics, with total search times ranging from **6 to 12 hours**.

Table 9: Empirical wall-clock time for representative tasks (Total 5-iteration search).

Task Name	Total Time (H:M:S)
AllegroHand	11:57:05
Ant	00:48:19
Anymal	01:46:43
FrankaCabinet	01:34:50
Quadcopter	00:46:03
ShadowHand	06:42:25
Catch Abreast	06:16:30
Door Close Outward	08:58:32
Kettle	12:09:01
Lift Underarm	08:18:31
Pen	07:55:03
Push Block	05:04:10
Re-orientation	05:46:50
Swing Cup	07:38:13

Analysis of Search Efficiency The results indicate that for simpler tasks, R^2E can discover near-optimal reward functions in under an hour. Even for the most challenging dexterous manipulation tasks in Bi-DexHands, the search concludes within half a day. Given that each iteration involves training 16 distinct policies, the average training time per candidate remains highly efficient due to our parallelized infrastructure. This demonstrates that R^2E is a practical and scalable reward for automated reward engineering in complex robotic domains.

E.3 Token Usage and Economic Cost Analysis

To rigorously evaluate the economic feasibility of deploying R^2E in large-scale experiments, we implemented a granular monitoring system to track the token consumption and associated API costs for each distinct task. Our cost analysis is based on the *Qwen3-Max* model utilizing the Batch Inference pricing tier. This tier is particularly suitable for evolutionary search processes where real-time latency is not a critical constraint, allowing for significant cost reductions compared to standard synchronous API calls.

Pricing Model and Accounting Method The reported costs are calculated based on the official batch inference rates for the China region (RMB), which are structured as follows:

- **Input Processing (Batch):** 0.0016 RMB per 1,000 tokens.
- **Output Generation (Batch):** 0.0064 RMB per 1,000 tokens.

The cumulative token count presented includes all prompt inputs (System, User, and Assistant context) and generated completions across all three evolutionary roles (*Explorer*, *Guardian*, *Evolver*) over the entire course of the 5-iteration search process. This ensures that the cost estimates reflect the complete end-to-end expense of synthesizing a finalized reward function.

Analysis of Consumption Patterns Table 10 summarizes the detailed token usage and estimated total cost. We observe a distinct correlation between task complexity and token consumption. Standard locomotion and single-arm manipulation tasks (e.g., *Ant*, *Quadcopter*, *FrankaCabinet*) typically consume between 90k to 120k total tokens. In contrast, high-dimensional bimanual manipulation tasks from the Bi-DexHands benchmark (e.g., *DoorCloseOutward*, *SwingCup*) exhibit a marked increase in usage, ranging from 170k to over 210k tokens. This approximate $2\times$ increase is attributed to the verbose context required to describe the dual-arm state spaces (often exceeding 200 dimensions) and the syntactic complexity involved in generating coordinated reward logic for multi-agent-like interactions.

Table 10: **Token Usage and Economic Cost Analysis.** The table details the cumulative prompt and completion tokens consumed over a full 5-iteration evolutionary search, along with the estimated total API cost in RMB.

Task Name	Prompt Tokens	Completion Tokens	Total Tokens	Est. Cost (RMB)
AllegroHand	76,118	31,464	107,582	0.32
Ant	73,182	38,234	111,416	0.36
Anymal	69,530	48,141	117,671	0.42
FrankaCabinet	60,717	34,121	94,838	0.32
Quadcopter	56,514	35,221	91,735	0.32
ShadowHand	75,419	23,899	99,318	0.27
CatchAbreast	66,860	49,772	116,632	0.43
DoorCloseOutward	127,790	73,231	201,021	0.67
Kettle	117,016	56,974	173,990	0.55
LiftUnderarm	83,272	48,480	131,752	0.44
Pen	124,741	72,784	197,525	0.67
PushBlock	114,172	49,836	164,008	0.50
ReOrientation	84,855	59,122	143,977	0.51
SwingCup	128,453	83,361	211,814	0.74

Economic Efficiency and Scalability The financial implications of R^2E are minimal when compared to traditional reward engineering workflows. As evidenced in Table 10, the total API cost for synthesizing a high-performing reward function remains under **1.0 RMB** (approximately \$0.14 USD) for every evaluated task. Even for the most intricate task, *SwingCup*, the cost caps at merely 0.74 RMB.

To put this into perspective, the computational cost of training the RL agent itself (GPU electricity and amortization) typically exceeds the LLM query cost by several orders of magnitude. Furthermore, compared to the labor cost of a domain expert—who might spend days iterating on reward coefficients—R^2E offers a cost reduction of several orders of magnitude. This extreme economic efficiency demonstrates that the R^2E framework is highly scalable and can be deployed to solve hundreds of diverse robotic tasks without incurring significant financial overhead.

E.4 Effect of LLM Backbone

We examine the sensitivity of reward evolution to different LLM backbones by substituting the generator used in both Eureka and R^2E, while keeping the same environments and evaluation protocol. Table 11 reports mean $\pm$ standard deviation over 5 seeds for two representative tasks.

Table 11: **Impact of LLM Backbone on Reward Evolution.** Performance is reported as mean ± standard deviation over 5 seeds.

Task	LLM Backbone	Eureka	R^2E
Ant	Qwen3-Coder-Flash	3.8851±0.0289	6.2114±0.6671
	Qwen3-Max	6.5876±0.2602	7.2639±0.4584
	GPT-5.4	6.3583±0.6967	7.2615±0.5690
ShadowHand	Qwen3-Coder-Flash	0.1626±0.0866	0.5577±0.3162
	Qwen3-Max	4.9272±1.4171	8.9603±1.4174
	GPT-5.4	7.8988±0.5859	9.6329±1.3460

From the results, we observe a clear trend that stronger LLM backbones consistently lead to improved performance. Notably, R^2E benefits more significantly from increased model capability, yielding larger performance gains as the backbone becomes more powerful. We attribute this to the fact that stronger models exhibit a better understanding of role specification and instruction following, which is crucial for R^2E's role-conditioned reward evolution process. This enhanced role awareness enables more coherent and effective reward design, ultimately leading to superior downstream policy performance.

E.5 LLM Backbone Diversity and Reward Homogenization

Different backbone models can follow different reasoning patterns, possibly because post-training imposes different optimization objectives and alignment strategies. To examine this difference, we compare *Qwen3.7-Max*, *DeepSeek-v4-pro*, and *GPT-5.6-sol-max* on *Ant* and *Pen*.

The three models approach reward design differently. *Qwen3.7-Max* first uses reliable signals already available in the environment and then adds reward terms based on those signals. *DeepSeek-v4-pro* first decomposes the task into intuitively reasonable behaviors and then combines the corresponding reward terms. *GPT-5.6-sol-max* first considers which reward designs are likely to cause training failures and then infers which combinations of reward terms are better aligned with the task. These observations suggest that using different models can reduce homogeneity among the sampled reward functions. They do not imply that switching the backbone across iterations within a single run will improve the final task performance.

Table 12 reports the final evaluation. *Ant* is measured by its task-specific performance metric, and *Pen* by success rate. In the mixed setting, the backbone is assigned by iteration: iterations 1 through 5 use *Qwen3.7-Max*, *DeepSeek-v4-pro*, *GPT-5.6-sol-max*, *Qwen3.7-Max*, and *DeepSeek-v4-pro*, respectively. Alternating among the three models does not outperform the strongest single backbone on either task. We conjecture that switching models can disrupt the coherence of the reasoning process across iterations.

Table 12: **Single and mixed LLM backbones.** *Ant* reports the task-specific performance metric, and *Pen* reports success rate. Results are mean ± standard deviation.

LLM Backbone	Ant	Pen
Qwen3.7-Max only	7.5604 ± 0.6164	0.9678 ± 0.0057
DeepSeek-v4-pro only	7.6408 ± 0.1465	0.9468 ± 0.0416
GPT-5.6-sol-max only	7.9866 ± 0.3364	0.9769 ± 0.0014
Mixed (alternating LLMs)	7.7542 ± 0.2541	0.9538 ± 0.0048

F Limitation

Limitation. Our R^2E method, like Eureka, REvolve, and RF-Agent, falls into the category of reward function design approaches that depend on iterative feedback from LLMs combined with repeated policy optimization. Consequently, the overall pipeline incurs substantial computational cost and time overhead, as each task involves multiple rounds of LLM interaction alongside several reinforcement learning training cycles. Although R^2E can deliver strong generation performance even when built on relatively small LLM backbones, it does not alleviate the need for numerous RL iterations in the outer-loop search. In future work, we aim to reduce the number of required RL training cycles while preserving the advantages of iterative reward refinement.

A further limitation of our approach is its reliance on a sufficiently strong base model. Since the method depends on accurate role understanding and nuanced feedback, weaker backbone models tend to exhibit degraded performance, which in turn negatively impacts the overall effectiveness of the system.

G Algorithm

The operational workflow of R^{2}E is summarized in Algorithm 1. It begins by generating an initial set of reward functions to explore the reward space and establish a diverse seed population. From the second iteration onward, candidates are evolved through the Explorer, Guardian, and Evolver roles under a dynamic scheduling strategy that shifts focus from exploration to refinement. Each candidate is evaluated via RL training, and best-performing rewards are stored in the Global Elite Pool to guide subsequent search. This process balances novelty and stability, mitigating instability and enabling effective evolution of reward functions.

Algorithm 1 R^{2}E: Role-driven Reward Evolutionary Framework

Require: Environment E, LLM M, total iterations T, budget per iteration N, pool size K

Initialize Global Elite Pool $\mathcal{B} \leftarrow \emptyset$
for $t = 1$ **to** T **do**
  if $t == 1$ **then**
    Generate N initial reward functions $\{r_i\}_{i=1}^{N}$ using M to broadly explore the reward space
  else
    Determine role allocation $\{N_{\text{Exp}}, N_{\text{Gua}}, N_{\text{Evo}}\}$ via dynamic scheduling strategy
    Explorer: Query r_{best} and $r_{rand} \in \mathcal{B}$ to generate N_{Exp} novel candidates
    Guardian: Query Top-1 reward $r_{best} \in \mathcal{B}$ to generate N_{Gua} stable variations
    Evolver: Query r_{best} and $r_{rand} \in \mathcal{B}$ to cross over and mutate N_{Evo} offspring
    Aggregate all generated reward functions $\{r_i\}_{i=1}^{N}$
  end if
  for each reward function r_i **do**
    Execute RL training process in E guided by r_i
    Evaluate task performance $\mathcal{J}(\pi_{r_i})$ via validation rollouts
    if $|\mathcal{B}| < K$ **then**
      Insert r_i into $\mathcal{B}$
    else if $\mathcal{J}(\pi_{r_i}) > \min_{r \in \mathcal{B}} \mathcal{J}(\pi_r)$ **then**
      Replace the lowest performing reward in $\mathcal{B}$ with r_i
    end if
  end for
end for
Return the top-performing reward function $r^* \in \mathcal{B}$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We accurately claim the main contributions and scope of our paper in the Abstract and Introduction section.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: Our paper does not include theoretical results, so this question is not applicable to our paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.

- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Appendix D contains all information required to reproduce our experimental results. Note that due to the inherent randomness of LLMs, exact numerical values cannot be perfectly reproduced, while the overall result range and comparative trends are reproducible.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The full code and experimental data are provided in our supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix D and Section 5.1 provide detailed information about the experimental setup.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide statistical results and illustrate the experimental variability in Section 5.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the detailed hardware and computing resource information in Appendix E.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work fully complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work focuses on reward function design for large language models in embodied scenarios. It belongs to foundational algorithmic research and has no direct negative societal impacts or foreseeable malicious applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: There are no such risks in our paper.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All publicly available datasets and open-source frameworks used in this work are properly cited, with their original licenses and usage terms respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Included with the supplementary material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No crowdsourcing or human-subject research is involved in this paper.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work involves no human-participant experiments, so IRB approval or equivalent ethical review is not required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Detailed usage of LLMs is provided in Section 4, Appendix D.2 and Appendix E.3.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.