Efficient Language-instructed Skill Acquisition via Reward-Policy Co-Evolution

Changxin Huang[1], **Yanbin Chang**[1], **Junfan Lin**[2*], **Junyang Liang**[1], **Runhao Zeng**[3], **Jianqiang Li**[1*]

[1]National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Artificial Intelligence Research Institute, Shenzhen MSU-BIT University, Shenzhen, China
huangchx@szu.edu.cn, changyanbin2023@email.szu.edu.cn, linjf@pcl.ac.cn,
liangjunyang2018@email.szu.edu.cn, zengrh@smbu.edu.cn, lijq@szu.edu.cn

Abstract

The ability to autonomously explore and resolve tasks with minimal human guidance is crucial for the self-development of embodied intelligence. Although reinforcement learning methods can largely ease human effort, it's challenging to design reward functions for real-world tasks, especially for high-dimensional robotic control, due to complex relationships among joints and tasks. Recent advancements large language models (LLMs) enable automatic reward function design. However, approaches evaluate reward functions by retraining policies from scratch placing an undue burden on the reward function, expecting it to be effective throughout the whole policy improvement process. We argue for a more practical strategy in robotic autonomy, focusing on refining existing policies with policy-dependent reward functions rather than a universal one. To this end, we propose a novel reward-policy co-evolution framework where the reward function and the learned policy benefit from each other's progressive on-the-fly improvements, resulting in more efficient and higher-performing skill acquisition. Specifically, the reward evolution process translates the robot's previous best reward function, descriptions of tasks and environment into text inputs. These inputs are used to query LLMs to generate a dynamic amount of reward function candidates, ensuring continuous improvement at each round of evolution. For policy evolution, our method generates new policy populations by hybridizing historically optimal and random policies. Through an improved Bayesian optimization, our approach efficiently and robustly identifies the most capable and plastic reward-policy combination, which then proceeds to the next round of co-evolution. Despite using less data, our approach demonstrates an average normalized improvement of 95.3% across various high-dimensional robotic skill learning tasks.

Introduction

As affordable robots are increasingly common, it becomes more and more crucial to develop algorithms that enable robots to understand commands, solve problems, and self-evolve, reducing the need for constant human supervision. Reinforcement learning (RL), which employs reward functions to decrease reliance on human supervision (Haarnoja et al. 2024), has been a triumph across various domains, including video games (Alonso et al. 2020) and the strategic
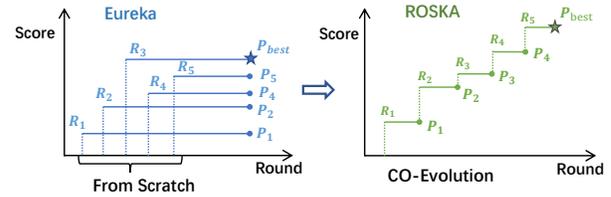
Figure 1: Comparison of main differences between our method and Eureka.

game of chess (Silver et al. 2018). Yet, when we shift our focus to the intricate realm of high-dimensional robot loco-motion (Radosavovic et al. 2024), the task of formulating a reward function for fundamental movement becomes notably complex. It's not just a matter of technical know-how; it's about navigating the unique intricacies of each robot's form and joint configuration (Andrychowicz et al. 2020).

In recent years, with the rapid growth of large language models (LLMs) (Minaee et al. 2024), numerous approaches have been proposed to enhance robots' ability to execute tasks based on human instructions. For instance, Google's SayCan approach integrates LLMs with robotic affordances, allowing robots to generate feasible task plans grounded in real-world actions (Brohan et al. 2023). Similarly, the Code as Policies method leverages LLMs to autonomously generate robot policy code from natural language commands, enabling robots to generalize to new instructions and environments without additional training (Liang et al. 2023). Among these, Euraka is a standout initiative, marking a pioneering step in the self-design of reward functions tailored to task instructions with the help of LLMs (Ma et al. 2024). They aim to develop a universally applicable reward function that can guide a policy with randomly initialized parameters to learn and perform tasks directed by language instructions.

Despite its success, Eureka is still confined by the traditional RL framework, which relies on one reward function to provide feedback throughout the policy improvement process. This can lead to inefficient and ineffective policy optimization. Firstly, each intermediate reward function necessitates learning a policy from scratch, as shown on the left of Fig.1. Secondly, finding a universal reward function is

non-trivial because it must be sufficiently comprehensive to consistently offer task-relevant feedback for different state-action-state transitions across the whole policy improvement process. This not only results in an enormous search space but also demands a more sophisticated prompt design for the LLM as the complexity of tasks increases. This orthogonal approach to improving reward functions and policies separately is costly and can impede real-world applications.

In our exploration, we've identified a transformative opportunity to refine the traditional RL model, gearing it towards greater efficiency and real-world practicality in the era of LLM. Recognizing the prowess of LLMs to craft reward functions for specific commands, we propose to harness this further: we aim to empower LLMs to autonomously adapt and refine the reward functions in conjunction with policy improvements. In this paper, we propose a novel **reward-policy co-evolution framework** for efficient language-instructed RObot SKill Acquisition (**ROSKA**), where the reward function and the policy co-evolve in tandem, rather than separately, exponentially speeding up the learning process, as sketched in the right of Fig.1.

Specifically, for the evolution of the reward function, we've crafted a cutting-edge LLM-driven mechanism for evolving reward functions, ensuring steady enhancement at each evolutionary phase. This is accomplished by dynamically expanding the population of reward function candidates generated by the LLM, using the historically top-performing reward function as a benchmark for this expansion. In regards to the policy evolution, to adapt the ongoing policy to become both capable and plastic to the new reward functions, we initiate with policy candidates by fusing the parameters of the previous best policy with a dash of randomized parameters in varying proportions. The top-performing fused policy will be selected for further optimization under their respective reward functions. To quickly identify the promising fused policies from countless possible fused policies, we've implemented Short-Cut Bayesian Optimization (SC-BO) for a faster policy evolution. SC-BO leverages the observation that different fused policies generally diverge after a few updates to conduct an early stop optimization. Additionally, SC-BO only uses a limited number of search points allowing for fast searches, and the dynamic population of the reward evolution will guarantee the continual refinement of policies. This method achieves superior results with fewer iterations compared to vanilla BO.

At the end of each reward-policy co-evolution cycle, the most effective reward-policy combination will initiate the subsequent round of reward-policy co-evolution.

Extensive experimental results demonstrate that our approach utilizes only 89% of the data and achieves an average normalized improvement of 95.3% across various high-dimensional robotic skill-learning tasks, highlighting its effectiveness in enhancing the adaptability and precision of robots in complex environments.

Related Work

Designing reward functions for robot. Designing reward functions for applying reinforcement learning to robotic

tasks has long been a significant challenge. Existing approaches can be broadly categorized into manually designed rewards and automated reward generation (Yu et al. 2023). Manually designed rewards rely heavily on extensive domain knowledge and experience (Booth et al. 2023).

Automated reward design includes Inverse Reinforcement Learning (IRL) (Pinto et al. 2017) and LLM-based reward generation (Ma et al. 2024). IRL is a data-driven approach that derives reward functions from demonstration data (Ziebart et al. 2008). However, IRL relies on high-quality demonstration data, which is often expensive to collect, particularly for robotic applications.

Recently, large language models (LLMs) have been employed to design reward functions by directly converting natural language instructions into rewards (Lin et al. 2022; Hu and Sadigh 2023), such as in "text2reward" (Xie et al. 2023) and "language2reward" (Yu et al. 2023). However, these methods typically require predefined reward templates or initial reward functions. To design reward functions from scratch, NVIDIA researchers introduced the Eureka framework, which employs a multi-round iterative process to design reward functions and uses RL to train robots for complex skills (Ma et al. 2024). The reward functions generated by Eureka must be trained to verify their effectiveness, with results fed back to the LLM for further refinement, leading to high training costs. In contrast, our approach fine-tunes a pre-trained policy on a new reward function, significantly enhancing data efficiency.

Leveraging pre-trained policies to enhance training efficiency in RL. The utilization of pre-trained policies to fine-tune models in new environments, thereby improving training efficiency, has proven effective in robotic tasks (Kumar et al. 2022; Walke et al. 2023). Common approaches include offline RL (Kumar et al. 2020) and meta RL (Wang et al. 2023). Offline RL algorithms develop robot control policies from pre-existing demonstration data or offline interaction datasets. These pre-trained policies can then be utilized during the online fine-tuning phase to adapt to novel tasks (Lee et al. 2022). Meta RL focuses on training policies across diverse tasks that can rapidly adapt to new ones (Arndt et al. 2020). We propose incorporating pre-trained policies into the LLM-based reward design process to accelerate training, avoiding the need to start from scratch. To align the pre-trained policy with the designed reward, we introduce a novel policy evolution method using Bayesian Optimization to determine the optimal inheritance ratio.

Preliminary

Reinforcement learning in robotic skill learning. Multi-joint robotic skill acquisition can be formulated as a Markov Decision Process (MDP) (Puterman 1990), where the robot interacts with the environment E. An MDP is represented as (S, A, P, R, γ), which includes the state space S, action space A, state transition probability function P, reward function R, and discount factor $\gamma \in (0, 1]$. At each time step t, the robot observes a state $s_t \in S$ and selects an action a_t according to the policy $\pi(s_t)$. The environment then transitions to the next state $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$, and the robot receives a reward $r_t = R(s_t, a_t, s_{t+1})$ and updates
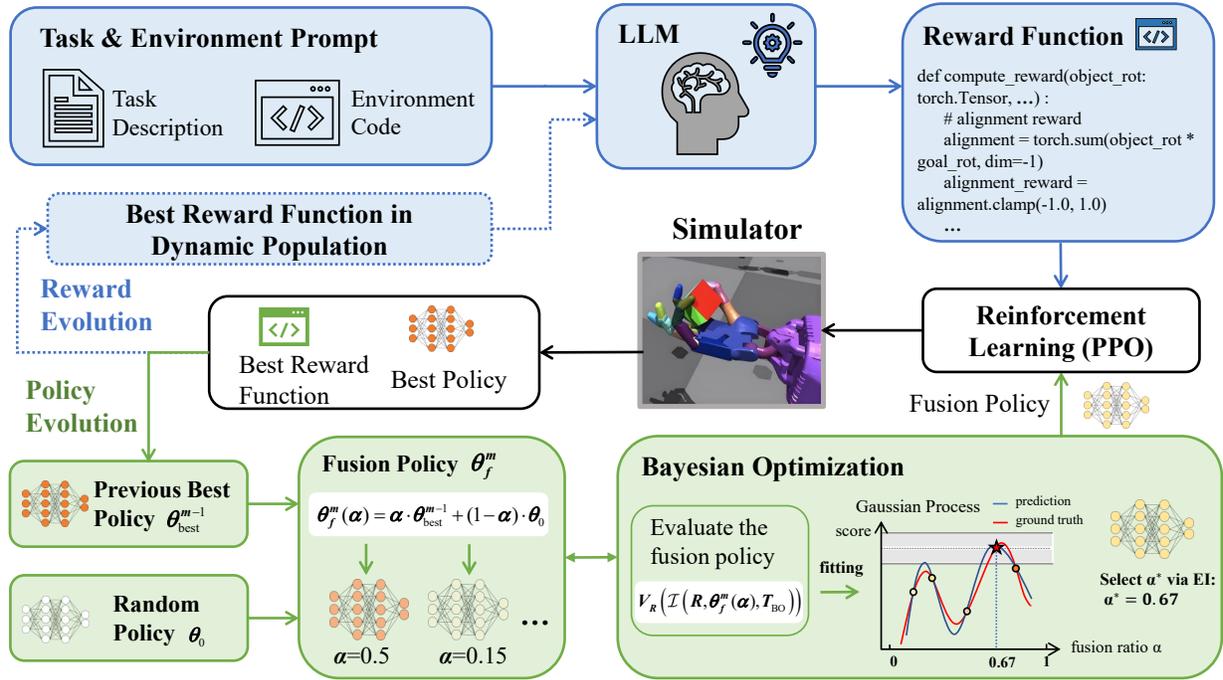
Figure 2: Overview of the proposed reward-policy co-evolutionary framework, illustrating the iterative refinement of reward functions and policies through mutual feedback between a large language model (LLM), reinforcement learning (PPO), and Bayesian optimization, enabling efficient and effective skill acquisition.

its state. The return for state s_t is defined as the cumulative γ-discounted reward: $\sum_{i=t}^{T} \gamma^{i-t} r_i$. Reinforcement learning aims to optimize the policy π by maximizing the expected return from the initial state. Formally, we define θ_p as the parameters of policy π after p updates, and

$$\theta_p = \mathcal{I}(R, \theta_0, p), \qquad (1)$$

where θ_0 represents the randomly-initialized parameters of policy π, and function $\mathcal{I}(R, \theta, p)$ stands for policy improvement process that the parameters θ are improved after p updates given the reward function R. To this end, the optimal policy parameters can be formulated as:

$$\theta^* = \mathcal{I}(R, \theta_0, \infty). \qquad (2)$$

language-instructed reward function generation. The design of the reward function is crucial in reinforcement learning and often relies on the experience of researchers and practitioners, involving iterative adjustments through trial and error. This is particularly challenging for multi-joint robotic skill acquisition, as it requires considering multiple factors, such as robot stability and task relevancy. To address the challenge of designing complex reward functions, a recent study, i.e., Eureka (Ma et al. 2024), utilizes Large Language Models (LLMs) to generate executable reward function code progressively. Specifically, given the task description I_d and the environment code I_e, the LLM generate reward functions by:

$$\mathbf{R}^n = \text{LLM}(I_d, I_e, R_{\text{best}}^{n-1}, V(\theta)), \qquad (3)$$

where $\mathbf{R}^n = [R_1^n, R_2^n, ..., R_K^n]$ represents the set of K reward functions generated by the LLM at round n, $V(\theta)$ stands for the return of the policy π_θ under the ground-truth sparse reward function R^* and $R_{best}^{n-1} \in \mathbf{R}^{n-1}$ is the most suitable reward code in last round:

$$R_{\text{best}}^{n-1} = \max_{R_k^{n-1} \in \mathbf{R}^{n-1}} V\left(\mathcal{I}(R_k^{n-1}, \theta_0, T_{\max})\right). \qquad (4)$$

Eq.(4) means that each generated reward functions are used to optimize a policy from scratch θ_0, and after $T_{\max}$ training epochs of improvement, the reward function whose policy obtains the best performance of the task is considered the most suitable reward function at this round.

Method

Overview

In this paper, we propose a novel reward-policy co-evolution framework for RObot SKill Acquisition (**ROSKA**), that enables robots to learn how to complete language-instructed tasks via automatic reward functions and policy co-evolution, significantly improving the efficiency and efficacy. As illustrated in Fig.2, ROSKA can be roughly divided into two distinctive yet mutually enhanced modules: **reward evolution** and **policy evolution**.

Briefly, the reward evolution module (as shown in the upper part of Fig. 2) prompts the LLM with the robot's task and environment descriptions to generate a dynamic set of reward functions. The most effective reward (evaluated after policy evolution) from this set is selected to proceed to

the next evolution cycle; as for policy evolution (lower part in Fig. 2, given a newly generated reward function, rather than starting from scratch, ROSKA builds on the previous round's best-performed policy to leverage its learned capability, blending its parameters with random noise to maintain plasticity. We use Bayesian Optimization with an early stop mechanism to find the optimal blending ratio that can balance between retaining learned skills and allowing for new learning. The policy with the best performance across all candidate rewards will proceed to the next round the policy evolution.

In the following sections, we will elaborate on each of these modules in detail.

Reward Evolution with Dynamic Population

In Eureka (Ma et al. 2024), within each round of reward searching, a set number of K reward functions are generated by a Large Language Model (LLM), as depicted in Eq.(3). Each reward function is then thoroughly tested by training a new policy, as shown in Eq.(4). However, we've discovered a potential performance degradation with this approach: if the size of candidate reward functions is small, there is a chance that no superior reward functions could be identified. Using a large size could mitigate this, but it could be costly. After all, interacting with an LLM isn't cheap, and only one reward function will be chosen, rendering the rest obsolete.

To address this, we have introduced an approach to dynamically adjust the size of the population of the reward functions $\mathbf{R}_{\mathrm{DP}}$ in our reward evolution process. This allows the size of the population to increase when a larger exploration is needed to witness an improvement. Formally,

$$\mathbf{R}_{\mathrm{DP}}^m = \mathrm{LLM}\left(I_d, I_e, R_{\mathrm{best}}^{m-1}, V(\theta)\right), \qquad (5)$$

where $\mathbf{R}_{\mathrm{DP}}^m = [R_1^m, R_2^m, ..., R_{\mathrm{DP}}^m]$. We use m to indicate the m-th DP-round to distinguish it from the concept of round mentioned previously.

To determine when to increase the size of $\mathbf{R}_{\mathrm{DP}}^m$, analogical to how world records in the Olympics inspire athletes to push their limits and excel, we use the top-performing reward function from the previous DP-round, i.e., R_{best}^{m-1} as a benchmark, and repeating the process of Eq.(5) to generate K reward functions each time until a reward function with better performance than R_{best}^{m-1} is witnessed. Interestingly, we found that with the same total amount of generated reward functions, our method dynamically allocates the number of query rounds at each DP-round, and can achieve a consistent improvement upon using the fixed size of candidates at each round. This necessitates the dynamic sizes during reward evolution to avoid performance degradation.

And to effectively and efficiently identify the best reward function from $\mathbf{R}_{\mathrm{DP}}^m$, instead of evaluating each reward function by training a policy from scratch like Eq.(4), we propose to evaluate them based the evolved policy which is modified from the best-performed policy with parameters $\theta_{\mathrm{best}}^{m-1}$ of the previous DP-round.

$$R_{\mathrm{best}}^m = \max_{R_k^m \in \mathbf{R}_{\mathrm{DP}}^m} V\left(\mathcal{I}_{\mathrm{evolve}}\left(R_k^m, \theta_{\mathrm{best}}^{m-1}, T_{\max}\right)\right), \quad (6)$$

where $\mathcal{I}_{\mathrm{evolve}}$ is the policy evolving process elaborated in the following section.

Policy Evolution via Bayesian Optimization

To fully leverage the knowledge from previously trained policies and enhance training efficiency, each DP-round reuses the parameters of the best-performed policy parameters from the previous DP-round. This means the policy does not learn from scratch given a new reward function, unlike traditional policy improvement formulated in Eq.(1) adopted by Eureka in Eq. (4), largely eliminates the initial learning process where a policy is not able to perform basic operations, significantly improving the sample efficiency of RL training.

Unfortunately, when LLM designs a reward function that's quite different from the previous one, simply copying the previous policy parameters could result in a policy with a slow convergence on the new reward function, or it might not converge at all, as noted in (Parisotto, Ba, and Salakhutdinov 2016). What's worse, if the policy has fully converged and its parameters are saturated without plasticity, finetuning on its parameters directly might cause slow improvements and require more training epochs than training from scratch, as found in (Dohare, Hernandez-Garcia, and Rahman 2023).

To this end, we propose a partial inheritance method, where the parameters of the policy selected from the previous DP-round are randomly corrupted by fusing with randomly initialized parameters, formulated as:

$$\theta_f^m(\alpha) = \alpha \cdot \theta_{\mathrm{best}}^{m-1} + (1 - \alpha) \cdot \theta_0, \qquad (7)$$

where $\theta_f^m(\alpha)$ represents the fused model parameters, and $0 \leq \alpha \leq 1$ denotes the fusion ratio. By fusing pretrained model parameters with random model parameters, we aim to leverage the accumulated knowledge from previous training to accelerate convergence under new reward functions, while also retaining model plasticity. By this definition, the policy improvement process with a new reward function R with p updates can be formulated as $\mathcal{I}(R, \theta_f^m(\alpha), p)$.

Bayesian Optimization for Searching Fusing Ratio. The key question is how to determine α. The simplest approach is to uniformly sample α values from [0, 1], obtaining multiple fused parameters through Eq.(7), and then validating the performance of each α value through $\mathcal{I}(R, \theta_f^m(\alpha), T_{\max})$. However, this exhaustive method is inefficient. To enhance the efficiency of searching for the optimal fusion ratio and reduce training costs, this paper employs a Bayesian Optimization (BO) method based on Gaussian processes (GP) to search for the optimal fusion ratio.

Specifically, we define the relationship between the RL policy performance score and the fusion ratio α as

$$s(\alpha; \theta_f^m(\alpha), T_{\mathrm{BO}}) = V\left(\mathcal{I}\left(R, \theta_f^m(\alpha), T_{\mathrm{BO}}\right)\right), \quad (8)$$

where T_{BO} denotes the number of updates before calculating $s(\alpha)$. BO assumes that the RL policy performance score s follows a multivariate normal distribution. To this, we use Gaussian processes and data samples $D = \{(\alpha_i, s_i)\}_{i=1}^n$ to construct the posterior distribution of the objective function.

Specifically, we first select several initial points of the fusion ratio $(\alpha_1, \alpha_2, \ldots, \alpha_i)$ and evaluate the corresponding model performance scores $(s_1, s_2, \ldots, s_i)$ using Eq.(8).

Subsequently, we use the initial fusion ratios and their corresponding performance scores to construct the predictive distribution of the objective function. When selecting new evaluation points in subsequent iterations, the Gaussian process model calculates the Expected Improvement (EI) (Astudillo and Frazier 2022) of the evaluation point and selects the point α_{i+1} that maximizes EI as the new evaluation point. The iteration process is carried out in J rounds.

Short-Cut Bayesian Optimization. In theory, a Gaussian Process (GP) with ample data $D = \{(\alpha_i, s_i)\}_{i=1}^n$ can precisely model policy performance and identify the best α. However, collecting too many samples is costly. Instead, we leverage the observation that policy performance generally diverges early in training, to apply BO with an early stop with a limited set of ample data, significantly speeding up the process, which we call Short-Cut Bayesian Optimization (SC-BO). Note that, even if α isn't optimal, the dynamic population ensures performance improvements continue.

Formally, the fusion ratio search by SC-BO, denoted as $\alpha_{\text{SC-BO}}$, w.r.t. to reward function R and policy with parameter θ is defined as:

$$\text{SC-BO}(R, \theta) = \arg\max_\alpha s(\alpha, \theta, T_{\text{BO}}). \tag{9}$$

Overall, our policy evolution process given the best policy from the previous DP-round $\theta_{\text{best}}^{m-1}$ and newly generated reward function R can be formulated as:

$$\mathcal{I}_{\text{evolve}}(R, \theta_{\text{best}}^{m-1}, T_{\text{max}}) = \mathcal{I}(R, \theta_f^m(\alpha_{\text{SC-BO}}), T_{\text{max}}). \tag{10}$$

Reward-Policy Co-Evolution

In this section, we will further break down the mechanics of our reward-policy co-evolution from the following aspects. **1) Evaluating the reward function:** In every DP-round, we identify the best new reward function by comparing it with the policy that won the last DP-round. This policy's reward is what we use to prompt the LLM for new rewards. The connection between the new reward and the existing policy is closer than with a brand-new policy, making this evaluation method more efficient and reflective of the new rewards' true performance. **2) Guiding the policy improvement:** The best reward from the previous round is then expanded upon by the LLM, generating a new set of reward functions. These new rewards are crafted to further refine the policy, nudging it closer to the optimal one. **3) Dynamic population as a filter:** Our dynamic population mechanism acts like a sieve for the reward-policy combination. Only those combinations that outshine the previous round's top performer survive to the next round. This ensures that only the better pairs can advance in the co-evolutionary process. **4) Efficient fusion policy selection:** Using SC-BO, we develop a policy that seamlessly builds on its predecessor's strengths and adapts well to new reward functions. This approach ensures efficient resource use and smooth progression, enabling continuous improvement of both rewards and policies with the same amount of interactions as Eureka.

Experiments

We conducted experimental evaluations of the proposed method within the Isaac Gym (Makoviychuk et al. 2021)
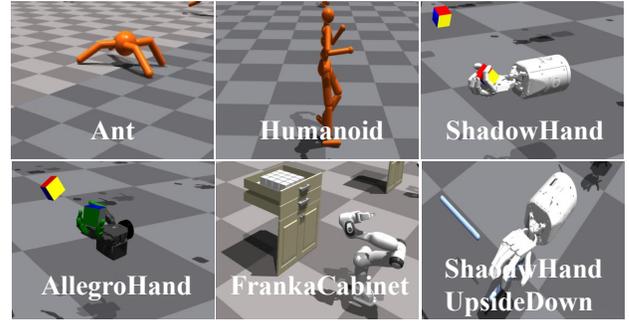

Figure 3: Illustrations of the six robot tasks in our experiment: Ant, Humanoid, ShadowHand, AllegroHand, Franka-Cabinet, and ShadowHandUpsideDown.

RL benchmark and performed comparative analyses against the sparse reward method, human-designed reward methods, and traditional LLM-designed reward function methods.

Experiment Settings

Environments and Tasks We validated our approach on six robotic tasks within Isaac Gym, including Ant, Humanoid, ShadowHand, AllegroHand, FrankaCabinet, and ShadowHandUpsideDown as shown in Fig. 3.

In our experiments, we employed the large language model GPT-4o to generate reward functions. Testing revealed that this model outperformed GPT-4 on most tasks (with the exception of the FrankaCabinet task) in terms of average performance. The RL method used to validate our proposed approach was Proximal Policy Optimization (PPO) (Schulman et al. 2017). In all experimental methods, the LLM conducted a total of $N = 5$ rounds of reward design for each robotic task, generating $K = 6$ reward functions for each round. As a reminder, a DP round could be viewed as the combination of several fixed-sized rounds, and in this section, we use the number of fixed-sized rounds for a clear and fair comparison. For the algorithm proposed in this paper, each reward function underwent policy evolution, where the Gaussian Process was initialized with fusion ratio points $\alpha_{\text{initial}} = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$. Each reward function underwent a total of $J = 12$ policy model evaluations, with $T_{\text{BO}} = 200$ for each evaluation. The other settings of our experiments can be found in the Appendix.

Baseline Methods To evaluate the performance of ROSKA, we mainly compare it with three baselines, i.e., sparse reward, human reward, and Eureka.
Sparse Reward (SR): SR refers to reward settings that express the task objective. These sparse reward functions are specifically defined in the settings described by Eureka (Ma et al. 2024).
Human Reward (HR): These reward functions are meticulously designed by researchers based on experience. Compared to sparse rewards, these rewards are more refined. Detailed definitions can be found in the environment settings within Isaac Gym (Makoviychuk et al. 2021).
Eureka: Eureka is a state-of-the-art algorithm that automat-

Methods	Task					
-	Ant	Humanoid	ShadowHand	AllegroHand	FrankaCabinet	ShadowHand-U
Sparse	6.59±1.44	5.12±0.49	0.06±0.039	0.06±0.02	0.0007±0.001	0.13±0.09
Human	10.35±0.12	6.93±1.38	6.00±1.02	11.57±0.53	0.10±0.05	14.86±6.36
Eureka	10.25±1.31	7.24±0.64	9.56±2.17	14.60±4.14	0.31±0.22	8.35±4.35
ROSKA-U	**12.52±1.03**	8.84±0.69	24.07±1.80	**26.80±2.17**	0.81±0.21	**23.72±4.96**
ROSKA	12.07±0.60	**9.10±1.06**	**24.34±2.84**	23.22±2.37	**0.85±0.19**	21.82±5.87

Table 1: MTS comparison across six tasks, presented as mean ± standard deviation of returns. Our method (ROSKA) consistently achieves superior performance across all tasks, outperforming other methods.

ically generates reward functions using LLM. It has shown outstanding performance across various robotic tasks.

ROSKA: ROSKA is the proposed reward-policy co-evolution framework.

ROSKA with Uniform Search (ROSKA-U): To evaluate the effectiveness of the policy evolution with SC-BO, we compared it with a uniform search method. While this method identifies a reasonably suitable fusion ratio, it requires an excessively large training sample size. For more details, see the Appendix.

Evaluation Metrics We employ Max Training Success (**MTS**) and Human Normalized Score (**HNS**) as the primary evaluation metrics in our experiments, consistent with Eureka (Ma et al. 2024). MTS reflects the average value of sparse rewards obtained during training, serving as a key indicator of model performance. HNS measures the algorithm's performance relative to human-designed reward functions. Given the scale differences in sparse reward metrics across tasks, we use the Human Normalized Score to facilitate performance comparison across different methods. The Human Normalized Score is calculated as follows:

$$HNS = \frac{MTS_{method} - MTS_{Sparse}}{|MTS_{Human} - MTS_{Sparse}|}, \quad (11)$$

where "Method," "Sparse," and "Human" represent the MTS values obtained from the method under evaluation, the sparse reward method, and the human reward method, respectively. In addition, we also used Total Training Samples (**TTS**) to evaluate the sample efficiency of each method.

Experimental Results and Analysis

Comparison to Baseline Methods We compared the MTS of the baselines and our method across six robotic tasks. As shown in Tab 1, our proposed method outperforms all baseline methods in all tasks. For instance, in the ShadowHand task, our method achieved a 154.6% improvement over the Eureka method, and in the ShadowHandUpside-Down task, it achieved a 184.07% improvement. The Eureka algorithm framework designs reward functions iteratively, with each round geared towards training from scratch using traditional RL methods. This approach cannot guarantee that each round of training will yield a higher score, as the reward functions designed by the LLM might be worse than those from previous rounds. In contrast, ROSKA inherits pretrained policy, effectively ensures an overall positive optimization trend. With the same number of reward
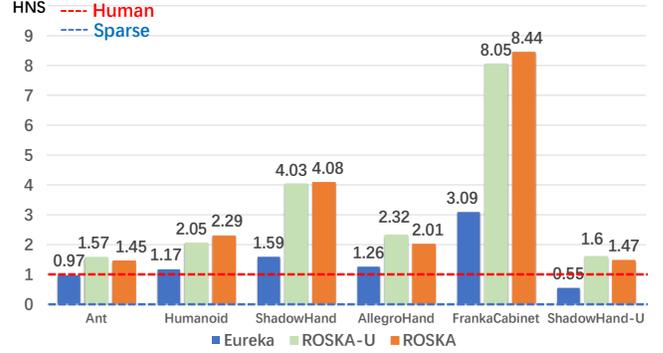

Figure 4: HNS comparison across six robotic tasks, demonstrating that our method consistently outperforms other methods, with substantial improvements across all tasks.

design rounds, our method significantly outperforms the Eureka algorithm. For example, in the AllegroHand task, our method outperforms Eureka by 83.56%. Compared to the ROSKA-U method, which is based on a uniform search for α, ROSKA achieved better results in Ant, AllegroHand, and ShadowHand-U tasks, while in other tasks, ROSKA-U performed slightly better. However, the training sample size for ROSKA-U is nearly 2.5 times that of ROSKA, which is further analyzed in ablation studies. This indicates that the ROSKA method can achieve performance comparable to that of the ROSKA-U method while using fewer samples.

We further compared the performance of each algorithm using the Human Normalized Score (HNS) metric, which more intuitively demonstrates the performance of our method relative to human-designed reward functions. HNS=1 indicates that the algorithm's performance is equivalent to that of human-designed rewards, and a higher HNS value signifies better algorithm performance. As shown in Fig. 4, our method surpasses the performance of expert-designed rewards in all six robotic tasks. Notably, in the ShadowHand and FrankaCabinet tasks, our method exceeds the performance of human-designed rewards by 4 times and 8 times, respectively, which is an extraordinary improvement. Compared to Eureka, our method achieved an average improvement of 95.3% on this metric.

To illustrate the reward-policy co-evolution mechanism, we visualized the training curves of ROSKA and Eureka across five rounds of reward function design, as shown in
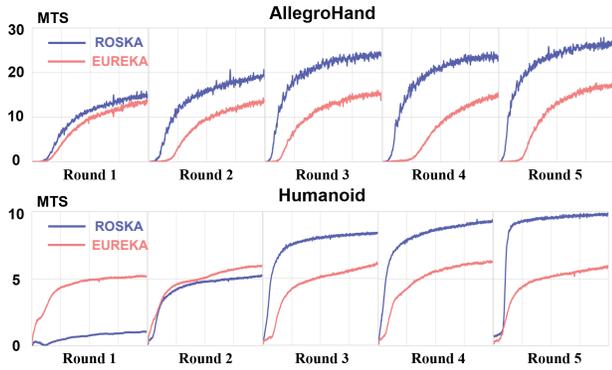
Figure 5: MTS comparison showing our method's steady improvement and higher scores over rounds, while Eureka struggles with stability. For details, see the appendix.

Method	Task		
-	Ant	Humanoid	ShadowHand
ROSKA-0%	10.25±1.31	7.24±0.64	9.56±2.17
ROSKA-50%	11.00±0.71	8.13±1.47	15.90±2.25
ROSKA-100%	11.06±1.13	6.24±3.76	21.58±5.10
ROSKA	**12.07±0.60**	**9.10±1.06**	**24.34±2.84**

Table 2: MTS Comparison of the SC-BO method with fixed fusion ratios, showing that SC-BO search method achieves the best performance across tasks.

Fig. 5. Results from the AllegroHand and Humanoid tasks demonstrate that, from the second round onward, ROSKA converges faster and achieves superior performance. This suggests that ROSKA effectively leverages pre-trained policy knowledge to enhance learning efficiency under new reward functions. In contrast, Eureka relies solely on reward function evolution, requires the policy to learn from scratch each round, leading to slower improvement. Notably, even when the LLM-generated reward in the first round of the Humanoid task was suboptimal, subsequent rounds saw rapid policy improvement, further validating the effectiveness of the reward-policy co-evolution mechanism.

Ablation Studies   The ablation study focuses on two key questions: First, the effect of the inherited pretrained model proportion α on policy evolution during reward design Second, the impact of training sample size on the final policy performance in the proposed Reward-Policy co-evolution method, evaluating the sample efficiency of our approach.

Effectiveness of SC-BO Search for Fusion Ratio. To evaluate the impact of inheriting the pretrained policy on the final model performance, we conducted experiments comparing the BO search method proposed in this paper with fixed fusion ratios. Fixed fusion ratio refers to a constant α value in each round during the design of rewards, representing the proportion of the historically optimal policy inherited. In our experiments, we selected $\alpha = [0\%, 50\%, 100\%]$ for three sets of experiments, denoted as ROSKA-0%, ROSKA-50%, and ROSKA-100%. When $\alpha = 0\%$, meaning that each round uses a randomly initialized policy for

Method	TTS	Task		
-	-	Ant	Humanoid	ShadowHand
Eureka	1	10.2±1.3	7.2±0.6	9.5±2.1
ROSKA-U	2.2	12.5±1.0	8.8±0.6	24.0±1.8
ROSKA	0.56	10.7±1.4	7.2±1.4	13.7±3.0
	0.74	10.9±0.7	7.1±1.2	14.4±3.7
	0.89	**12.0±0.6**	**9.1±1.0**	**24.3±2.8**

Table 3: MTS across tasks with varying TTS, presented as mean ± standard deviation. The Eureka method's TTS is used as the baseline (TTS = 1), with other methods' TTS expressed as proportions.

training. In Tab. 2, ROSKA-0% underperforms other methods that inherit pretrained parameters in the Ant and ShadowHand tasks, indicating that it's beneficial to inherit pretrained knowledge. Overall, ROSKA performed the best across all tasks, showing at least a 9% improvement over other methods with fixed fusion ratios. This result demonstrates that a reasonable inheritance ratio can significantly enhance the final performance of the policy.

Impact of Training Sample Size on Final Policy Performance. To validate the Sample Efficiency of the SC-BO method proposed in this paper, we evaluated the results obtained using different sample sizes. As shown in Tab. 3, even as the sample size decreases, our method continues to achieve good results. Notably, when using only 56% of the sample size required by the Eureka method, our approach still yields competitive results. For the uniform search method, although it can achieve good results, the required training sample size is extremely large. Our SC-BO-based method achieves comparable performance while using only 40% of the samples required by the uniform search method, significantly reducing the training cost. This indicates that the SC-BO method can efficiently find an optimal fusion ratio, enabling effective inheritance of pretrained policy knowledge and promoting rapid convergence of the policy under new reward functions.

Conclusion

In conclusion, our integration of large language models (LLMs) with reinforcement learning (RL) through the ROSKA framework has marked a significant leap in resolving language-instructed robotic tasks. ROSKA transcends traditional RL by enabling a co-evolution of reward functions and policies, synchronized to enhance each other. This symbiotic advancement optimizes the efficiency and effectiveness of robotic learning from language instructions. Extensive experiments witnessed an average improvement of 95.3% across complex robotic tasks with fewer samples, confirming the framework's potency. It highlights the capability of ROSKA to bolster robotic adaptability and autonomy, advancing the frontier of autonomous robotics.

Acknowledgements

References

Alonso, E.; Peter, M.; Goumard, D.; and Romoff, J. 2020. Deep reinforcement learning for navigation in AAA video games. *arXiv preprint arXiv:2011.04764*.

Andrychowicz, O. M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. 2020. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1): 3–20.

Arndt, K.; Hazara, M.; Ghadirzadeh, A.; and Kyrki, V. 2020. Meta reinforcement learning for sim-to-real domain adaptation. In *2020 IEEE International Conference on Robotics and Automation*, 2725–2731. IEEE.

Astudillo, R.; and Frazier, P. I. 2022. Thinking inside the box: A tutorial on grey-box Bayesian optimization. *CoRR*, abs/2201.00272.

Booth, S.; Knox, W. B.; Shah, J.; Niekum, S.; Stone, P.; and Allievi, A. 2023. The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 5920–5929.

Brohan, A.; Chebotar, Y.; Finn, C.; Hausman, K.; Herzog, A.; Ho, D.; Ibarz, J.; Irpan, A.; Jang, E.; Julian, R.; et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, 287–318. PMLR.

Dohare, S.; Hernandez-Garcia, J. F.; and Rahman, P. 2023. Maintaining Plasticity in Deep Continual Learning. *CoRR*, abs/2306.13812.

Haarnoja, T.; Moran, B.; Lever, G.; Huang, S. H.; Tirumala, D.; Humplik, J.; Wulfmeier, M.; Tunyasuvunakool, S.; Siegel, N. Y.; Hafner, R.; et al. 2024. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89): eadi8022.

Hu, H.; and Sadigh, D. 2023. Language instructed reinforcement learning for human-ai coordination. In *International Conference on Machine Learning*, 13584–13598. PMLR.

Kumar, A.; Singh, A.; Ebert, F.; Yang, Y.; Finn, C.; and Levine, S. 2022. Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials. *CoRR*, abs/2210.05178.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191.

Lee, S.; Seo, Y.; Lee, K.; Abbeel, P.; and Shin, J. 2022. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, 1702–1712. PMLR.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation*, 9493–9500. IEEE.

Lin, J.; Fried, D.; Klein, D.; and Dragan, A. 2022. Inferring rewards from language in context. *arXiv preprint arXiv:2204.02515*.

Ma, Y. J.; Liang, W.; Wang, G.; Huang, D.; Bastani, O.; Jayaraman, D.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2024. Eureka: Human-Level Reward Design via Coding Large Language Models. In *The Twelfth International Conference on Learning Representations, ICLR 2024*.

Makoviychuk, V.; Wawrzyniak, L.; Guo, Y.; Lu, M.; Storey, K.; Macklin, M.; Hoeller, D.; Rudin, N.; Allshire, A.; Handa, A.; et al. 2021. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*.

Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; and Gao, J. 2024. Large Language Models: A Survey. *CoRR*, abs/2402.06196.

Parisotto, E.; Ba, L. J.; and Salakhutdinov, R. 2016. Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016*.

Pinto, L.; Davidson, J.; Sukthankar, R.; and Gupta, A. 2017. Robust adversarial reinforcement learning. In *International conference on machine learning*, 2817–2826. PMLR.

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

Radosavovic, I.; Xiao, T.; Zhang, B.; Darrell, T.; Malik, J.; and Sreenath, K. 2024. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9(89): eadi9579.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Walke, H. R.; Yang, J. H.; Yu, A.; Kumar, A.; Orbik, J.; Singh, A.; and Levine, S. 2023. Don't start from scratch: Leveraging prior data to automate robotic reinforcement

learning. In *Conference on Robot Learning*, 1652–1662. PMLR.

Wang, H.; Liu, Z.; Han, Z.; Wu, Y.; and Liu, D. 2023. Rapid Adaptation for Active Pantograph Control in High-Speed Railway via Deep Meta Reinforcement Learning. *IEEE Transactions on Cybernetics*.

Xie, T.; Zhao, S.; Wu, C. H.; Liu, Y.; Luo, Q.; Zhong, V.; Yang, Y.; and Yu, T. 2023. Text2Reward: Automated Dense Reward Function Generation for Reinforcement Learning. *CoRR*, abs/2309.11489.

Yu, W.; Gileadi, N.; Fu, C.; Kirmani, S.; Lee, K.; Arenas, M. G.; and Chiang, H. L. 2023. Language to Rewards for Robotic Skill Synthesis. In *Conference on Robot Learning, CoRL 2023*, volume 229 of *Proceedings of Machine Learning Research*, 374–404.

Ziebart, B. D.; Maas, A. L.; Bagnell, J. A.; Dey, A. K.; et al. 2008. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, 1433–1438. Chicago, IL, USA.