Evolutionary Reward Design and Optimization with Multimodal Large Language Models

Ali Emre Narin
Kabatas Erkek High School
aliemre2024@gmail.com

Abstract

Designing reward functions is a pivotal yet challenging task for Reinforcement Learning (RL) practices, often demanding domain expertise and substantial effort. Recent studies have explored the utilization of Large Language Models (LLMs) to generate reward functions via evolutionary search techniques (Ma et al., 2023). However, these approaches overlook the potential of multimodal information, such as images and videos. In particular, prior methods predominantly rely on numerical feedback from the RL environment for doing evolution, neglecting the incorporation of visual data obtained during training. This study introduces a novel approach by employing Multimodal Large Language Models (MLLMs) to craft reward functions tailored for various RL tasks. The methodology involves providing MLLM with the RL environment's code alongside its image as context and task information to generate reward candidates. Then, the chosen agent undergoes training, and the numerical feedback from the environment, along with the recorded video of the top-performing policy, is provided as feedback to the MLLM. By employing an iterative feedback mechanism through evolutionary search, MLLM consistently refines the reward function to maximize accuracy. Testing on two different agents points to the preeminence of our approach over previous methodology, which themselves outperformed 83% (Ma et al., 2023) of reward functions designed by human experts.

1 Introduction

Large Language Models (LLMs) have shown remarkable success in distinct tasks. State-of-the-art models such as Gemini (Anil et al., 2023), Palm (Chowdhery et al., 2023), and GPT-4 (OpenAI et al., 2023) have achieved results comparable to human experts on different benchmarks. In this paper, we are specifically interested in their capabilities in designing Reward functions for Reinforcement Learning practices. Recent studies have shown that GPT-4 can autonomously generate reward functions for multiple agents in IsaacGYM by taking the environment code as context and employing evolutionary search (Ma et al., 2023). Impressively, it achieved results similar to and sometimes even better than those of human experts.

This result is very important for two reasons: firstly, the task of designing effective reward functions is notoriously challenging and time-consuming, and this approach streamlines the process by creating an end-to-end pipeline; secondly, by requiring no additional task-specific modifications, it showcases the generalization capabilities of evolutionary search on reward design.

However, a significant shortcoming of this approach, and LLMs in general, is that they can only operate on textual and numerical data. In contrast, when designing reinforcement learning strategies, human experts often leverage visual data to gain a deeper understanding of the problems that can be solved and improvements that can be made. It is our hypothesis that incorporating visual data could provide the model with enhanced comprehension, thus leading to improved accuracy.

We introduce EROM: "**E**volutionary **R**eward Design and **O**ptimization with **M**ultimodal Large Language Models (MLLMs)" method as a novel way to generate reward functions. In the EROM method, we utilize MLLMs' zero-shot coding abilities to generate reward functions. First, we provide the MLLM with the environment as context by providing the source code; then, we give it the description of the task, guidelines for reward function generation, and an image of the idle agent. After it generates the first iteration of the reward function, we provide feedback from the environment both numerically and visually by providing the video of the agent . Using evolutionary search, it generates a better set of reward functions, and this process iteratively continues.

Proceedings of the 3rd Workshop on Advances in Language and Vision Research, pages 202–208

Our contributions with the EROM method are as follows:

1. To the best of our knowledge, this is the first work that tests the MLLMs' abilities on reward function generation using evolutionary search.

2. We show that capturing the video (or image of an idle agent) of the top-performing policy and providing it to the MLLMs as feedback helps the performance, compared to just providing textual reflection.

3. By enhancing the qualities of an autonomous method that outperformed 83% human experts, we contribute to the advancement of autonomous reward design techniques without introducing significant computational cost or expenses.

Due to budget limitations, we mostly aimed to show a proof-of-concept of our approach. All the contributions listed above held true for our tests, but without more experiments, the (2) and (3)' rd contributions above should be approached tentatively.

2 Background

2.1 Reward Design

Reward design plays a pivotal role in reinforcement learning, where a well-crafted reward function is instrumental in achieving optimal outcomes. It guides agents toward actions aligned with the desired outcomes. Specifically, they provide positive feedback for actions conducive to achieving specific goals, while also providing negative feedback for actions that lack purpose or have a detrimental impact on the situation.

2.2 Evolutionary Search with LLMs

Evolutionary search algorithms, drawing inspiration from biological evolution, involve the generation of outputs by a generator, such as a LLM (Lehman et al., 2024). The generated outputs undergo evaluation, leading to feedback that informs subsequent iterations of output generation. This iterative process includes the generation of outliers, thereby mitigating the risk of the algorithm converging to a local optimum.

A recent study demonstrated notable success in leveraging Evolution with LLMs for the design of reward functions, incorporating textual feedback and information from the environment (Ma et al., 2023). In the present research, we extend this approach by introducing an additional modality of feedback—visual feedback—into the evolutionary process.

3 Methods

3.1 Environment as Context

The model needs to have a comprehensive understanding of the environment to generate a task-specific reward design for that environment. To achieve this, we give the environment source code as context to the model (Ma et al., 2023). This helps because providing the environment code gives the MLLM essential information about the variables used in the environment code and in what format we expect an output. Additionally, we augment the contextual information by presenting the MLLM with visual representations of the environment and agent. We believe this helps MLLM better understand the environment's visual cues and agent characteristics.

3.2 Evolutionary Search

We employ Evolutionary search for the iterative refinement of reward design. Initially, the model generates random samples of reward candidates, which are then evaluated on the task, and the top performer is selected. Subsequently, both reward feedback and the top performers are collected and fed back into the model for further enhancement. This iterative process is crucial, as evidenced by studies on LLMs demonstrating their capacity for self-improvement over time (Madaan et al., 2023). Moreover, this approach aligns with human intuition, as trial-and-error is a common strategy employed in the design of reward functions (Booth et al., 2023).

3.3 Reward Reflection

Previous studies utilizing LLMs to generate reward samples have primarily relied on textual feedback provided by the environment for evolutionary search (Ma et al., 2023). However, capturing the visual behavior of an agent can also yield valuable insights into necessary adaptations. For instance, visual feedback can aid in identifying instances of reward hacking or pinpointing areas where the agent is not performing as intended. To address this, following the initial iteration of reward sampling, each reward function is individually tested,

and both textual feedback from the environment and video recordings of the agent's performance are collected. Subsequently, for the subsequent iteration of evolution, the MLLM is provided with the code of the best-performing reward function, along with its numerical and video feedback gathered during training. The MLLM then reasons over this information to iteratively design improved reward functions. Through this process of reward reflection, the accuracy of designed rewards consistently improves, leading to notable outcomes in our experiments.

4 Experiments

4.1 Baselines

4.1.1 Environment

IsaacGYM (Makoviychuk et al., 2021) is a GPU-Accelerated Physics Simulation for robotics tasks. It enables hundreds of trainings to run at the same time, thus making it faster to conduct experiments. Also, we can capture videos during training, which is a prerequisite for our experiment. We picked humanoid and ant agents on two different tasks for our experiments on this simulator. The reason for selecting these agents was the GPU memory limit of our hardware.

4.1.2 Multimodal Large Language Model

GPT-4V(Vision) (OpenAI et al., 2023) is a MLLM that can take both visual and textual input. Its multimodal capabilities will allow it to reason over videos and images, and its natural language and programming capabilities will allow it to understand tasks and generate reward functions as Python codes, making it suitable to use in our experiments.

4.1.3 Eureka Method

Evolution-driven Universal Reward Kit for Agents (Eureka) (Ma et al., 2023) is a method that inspired us and the method that we built upon. The Eureka method involves providing the environment source code as context, evolutionary search to improve rewards, and using reward reflection. The only difference we made in our method is that we added visuals to the feedback loop and the environment as context part. We used very similar prompts to those of Eureka, with only minor changes indicating to the MLMM that we have added visuals. Also, Eureka has been shown to outperform 83% of human-expert-designed reward functions, which makes being able to outperform it a remarkable achievement.

4.2 Experimental Setup

We conducted three different tests to evaluate the effectiveness of our approach. Following the experiments originally described in the Eureka paper, we ran both EROM and Eureka for five iterations, generating 8 samples of reward function codes in each iteration. Due to the stochastic nature of MLLMs, when none of the codes worked in the first iteration, we reran it until at least one worked, resulting in guaranteed four rounds of feedback. We refer to this as "general testing" in the results subsection of our research.

We separately assessed the importance of providing an image of an agent in the first generation. We ran EROM and Eureka for one iteration, generating 32 samples. We have increased the sample size to have more examples to lower the chance factors that could effect results. We refer to this as "Image Testing" in the results subsection of our research.

We also separately assessed the importance of providing video during the feedback loop by providing the MLLM with the same reward codes generated in another iteration: one with only numerical feedback and the other with video feedback alongside numerical feedback. We generated 32 samples for both methods and compared them. We refer to this as "Video Testing" in the results subsection of our research.

Unless otherwise specified, when making experiments with EROM method, we provided the MLLM with a one-minute video of the agents training on the best policy generated during the training process (divided into 200 frames due to the context length of GPT-4V). In the reward sampling process, we trained the ant agent for 1500 epochs and the humanoid agent for 1000 epochs. In each training, the environment size was set to default for both agents. Each reward that achieved the best success rate in the initial training process was chosen to seed the next generation. We refer to the success rate obtained by reward functions in the initial iteration as "training-success" in the rest of the research. We evaluated the final best reward by retraining it over 5 different seeds and taking the average. We refer to this average as "average success."

4.3 Results

All the "average-success" results can be found in Table 1. Firstly, we observed that our method per-

Table 1: Average Success Rates

Test Type	Ant-EROM	Ant-Eureka	Humanoid-EROM	Humanoid-Eureka
General Testing	$7.27\|0.36\sigma$	$3.68\|0.71\sigma$	$5.26\|0.29\sigma$	$4.21\|0.53\sigma$
Video Testing	$6.13\|0.95\sigma$	$3.38\|0.39\sigma$	$5.42\|0.27\sigma$	$4.81\|0.70\sigma$
Image Testing	$6.38\|1.89\sigma$	$1.76\|0.87\sigma$	$3.17\|0.30\sigma$	$5.33\|0.39\sigma$

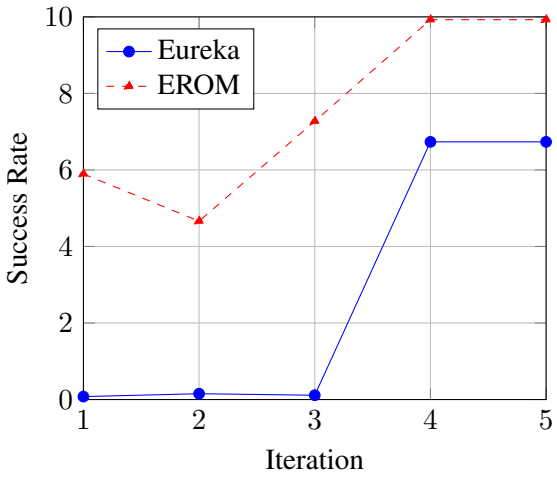

Figure 1: Comparison of success rates in General Testing on Ant agent.

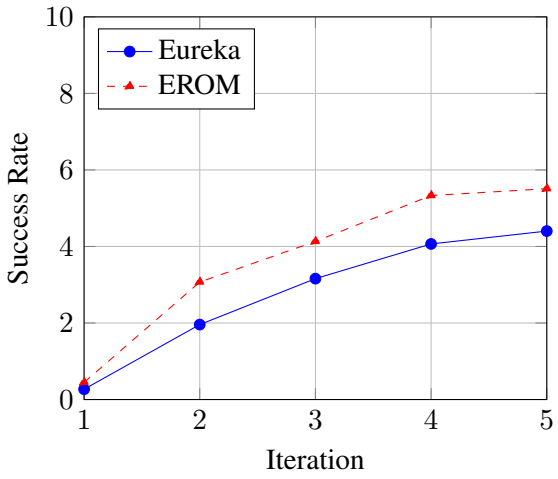

Figure 2: Comparison of success rates in General Testing on Humanoid agent.

formed better on general testing, where we ran both codes for 5 iterations with 8 samples generated in each iteration. On ant and humanoid agents, EROM achieved an average-success rate of 7.27 and 5.26, while Eureka achieved an average-success rate of 3.68 and 4.21, respectively. We have also plotted the difference between EROM and Eureka over the "training-success" of each iteration on Fig. 1, 2. These graphs effectively demonstrate the effectiveness of evolutionary search for both methods, as well as the value of video feedback and providing the image of the agent.

Secondly, to test the importance of providing the image of an agent in the first generation, we generated 32 samples using each method to increase the sample size and obtain a better average. As shown in Table 1, providing an image has shown to increase the average success rate for the ant agent, but not for the humanoid agent.

Lastly, by seeding the MLLM with the same reward functions and reward reflection, one with video and the other with only numerical feedback, we generated 32 samples with each method. We observed that providing the video also improved the average success for both of the agents.

5 Conclusion and Discussion

Designing effective reward functions is a laborious task that requires expertise and time. Recent researchers have sought to address this problem by utilizing Large Language Models (LLMs) to generate reward functions by taking the environment as context, employing evolutionary search, and utilizing reward reflection (Ma et al., 2023). However, they have only used numerical feedback and textual information for reward sampling and the reward reflection process. In this work, we address this limitation by incorporating videos of agents in training and their idle images into the evolutionary process with the help of Multimodal Large Language Models (MLLMs). Our aim is to enhance the success rate of previous methodology, which have already outperformed 83% (Ma et al., 2023) of human experts in their focused tasks. Experiments conducted with two agents across two distinct tasks have indicated that our approach is more effective than solely utilizing textual information.

Limitations

Since we utilized GPT-4V (OpenAI et al., 2023) in our experiments, results largely depend on its capabilities. Alongside that, the real-life applications of our method might not be as successful as in online simulation environments because of the complexity of the real world that is superficially present in simulations.

Another limitation of our work was that, due to the lack of GPU memory, we could only make tests on two agents in IsaacGYM. An experiment on more agents and different environments would better show our approach's generalization capabilities and effectiveness.

References

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, and ... Oriol Vinyals. 2023. Gemini: A family of highly capable multimodal models.

Serena Booth, W. Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. 2023. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(5):5920–5929.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran ..., and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Joel Lehman, Jonathan Gordon, Shawn Jain, Cathy Yeh, Kenneth Stanley, and Kamal Ndousse. 2024. *Evolution Through Large Models*, pages 331–366.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac gym: High performance gpu-based physics simulation for robot learning.

OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, and ... Barret Zoph. 2023. Gpt-4 technical report.

A Prompts

In this subsection, we provide the prompts used in our research. We have used the same prompts used in (Ma et al., 2023), with marginal changes regarding visuals.

```
The Python environment is
↪ {task_obs_code_string}. Write a
↪ reward function for the following
↪ task: {task_description}.
Here is an image of the agent.
↪ Carefully analyze it for better
↪ understanding.
<img src="{image_src}"
↪ alt="{image_alt}">
```

Figure 3: User Prompt

```
You are a reward engineer trying to
↪ write reward functions to solve
↪ reinforcement learning tasks as
↪ effective as possible.
Your goal is to write a 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.
```

Figure 4: System Prompt

```
We trained a RL policy using the
↪ provided reward function code
↪ and tracked the values of the
↪ individual components in the
↪ reward function as well as
↪ global policy metrics such as
↪ success rates and episode
↪ lengths after every
↪ {epoch_freq} epochs and the
↪ maximum, mean, minimum values
↪ encountered:
{Reward Reflection}
Please carefully analyze the policy
↪ feedback and provide a new,
↪ improved reward function that
↪ can better solve the task. Some
↪ helpful tips for analyzing the
↪ policy feedback:
    (1) If the success rates are
    ↪ always near zero, then you
    ↪ must rewrite the entire
    ↪ reward function
    (2) If the values for a certain
    ↪ reward component are near
    ↪ identical throughout, then
    ↪ this means RL is not able to
    ↪ optimize this component as
    ↪ it is written. You may
    ↪ consider
        (a) Changing its scale or
        ↪ the value of its
        ↪ temperature parameter
        (b) Re-writing the reward
        ↪ component
        (c) Discarding the reward
        ↪ component
    (3) If some reward components'
    ↪ magnitude is significantly
    ↪ larger, then you must
    ↪ re-scale its value to a
    ↪ proper range
Please analyze each existing reward
↪ component in the suggested
↪ manner above first, and then
↪ write the reward function code.
```

Figure 5: Feedback Prompt

B Computational Resources and Additional Expenses

We utilized an RTX 2060 6GB graphics card to execute all experiments. None of the experiments exceeded a runtime of 16 hours. We could only train one policy at a time for the humanoid agent, while two for the ant agent. The total cost of GPT-4V(ision) API calls, to run all the experiments, amounted to approximately $40.

C Task Details

In this section, we provide task details. For task details, we follow the structure from (Ma et al., 2023). We provide the task description, environment, observation and action dimensions, and the task fitness function F.

Table 2: Task Details and Descriptions

Environment	**Obs. Dim.**	**Act Dim.**	**Task Description**
Ant	60	8	To make the ant run forward as fast as possible (Fitness Function: $cur\_dist - prev\_dist$)
Humanoid	108	21	To make the humanoid run as fast as possible (Fitness Function: $cur\_dist - prev\_dist$)