VIRAL: Vision-grounded Integration for Reward design And Learning

Valentin Cuzin-Rambaud[a], **Emilien Komlenovic**[a] **and Alexandre Faure**[a]

[a]University Claude Bernard Lyon 1, France

Abstract. The issue of alignment between humans and machines is one of the most pressing concerns today. The goal of reinforcement learning is to maximize a reward function, which makes it susceptible to the consequences of a poorly designed reward function. Recent research has shown that large language models (LLMs) for reward generation can outperform human performance in this area. Thus, we propose VIRAL, a program for generating reward functions through the use of open-source, multi-modal LLMs. VIRAL autonomously generates and refines reward functions for a given environment with a specific prompt. VIRAL offers several features, including: generation from a simple prompt, generation from an annotated image, automatic improvement of the function with or without human feedback, and a description generated by a video LLM to explain the agent's policy in the video. We measure our method in five Gymnasium environments, enabling rapid learning of new behaviors and creating reward functions aligned with user intent. Documentation of the project is available here: *https://viral-ucbl1.github.io/*

1 Introduction

Creating effective reward functions is one of the hardest parts of working with reinforcement learning (RL). A poorly designed reward can send an agent down the wrong path, while a well-crafted one can speed up learning and guide the agent towards behaviors that match our expectations. The problem is that designing good rewards often takes a lot of manual effort and expertise, especially in robotic environments.

Thanks to their effectiveness in addressing a variety of problems, large language models (LLMs) are being utilized in robotics, particularly for RL. One of the first approaches [2], using natural language processing (NLP) techniques — such as RNNs and word encoding — to construct reward signals, demonstrated significant effectiveness in Atari games. An early method [4] that employed LLMs for reward generation used GPT-3 as a binary reward signal, but this approach can only be applied to a limited number of environments. More recently [7, 12, 14], the use of GPT-4 to generate code for reward functions has shown its competitiveness and its ability to produce effective reward functions.

The use of open-source LLMs with fewer parameters [3] is seemingly becoming competitive with GPT-4 and other large models. This trend is helping democratize the local use of LLMs or their deployment on small servers. LLMs using vision (LVLMs) [6, 8] that accept images allow users to communicate in ways beyond just text input. The combination of text and image input clarifies the user's intent, enabling the LLM to better grasp the meaning of the request. Finally, while still recent, LVLMs that use video as user input have shown promising results [5]. These LVLMs can describe a video, particularly the movements of objects within the scene.

In this paper, we present VIRAL: **V**ision-grounded **I**ntegration for **R**eward design **A**nd **L**earning. VIRAL is a framework to design rewards functions from simple users prompts.

The contributions presented in this paper are:

- A new way to design reward functions using simple natural language input, or annotated images.
- An automated pipeline to refine reward functions, augmented with human feedback or video description from a LVLM.
- A flexible, modular setup that can scale and adapt to different RL problems based on Gymnasium[13].
- A competitive approach that highlights the power of lighter LLMs.
- Quick inference achieved through multiprocessing.

Inspired by the state of the art, particularly by these latest papers[7, 14], we distinguish ourselves in several ways.

For the user's description of the environment, EUREKA[7] uses the environment's code, while Text2Reward[14] employs an abstraction of the environment through Pydantic class definitions. We have chosen to describe the environment solely through its observations, as outlined in the Gymnasium documentation. This approach enables the LLM to capture the necessary information for coherent generation while simplifying the implementation for the user.

Furthermore, we introduce two innovative vision features: the first is the use of annotated images, such as in Figure 3, to illustrate the goal for the first generation through drawings, and the second is a video description of the learned policy within a self-refining loop.

This paper is structured to detail the inner workings of VIRAL in Section 2, present our results in Section 3, and conclude the discussion in Section 4.

2 Architecture

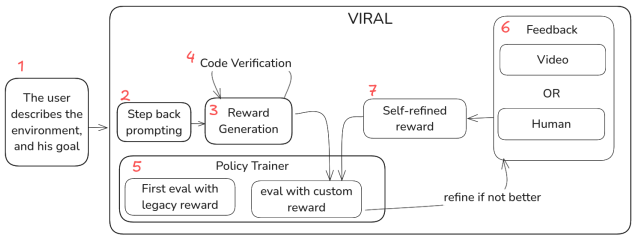

Figure 1. The VIRAL pipeline. The system begins with an environment description and iteratively generates and refines reward functions.

2.1 User Input Parameters

To effectively use our framework, users are required to provide a set of specific inputs (see Figure 1.1). The first input is the environment, any environment compatible with Gymnasium can be seamlessly integrated with VIRAL. The user constructs a prompt in JSON format, which includes the objective to be achieved and a detailed description of the environmental observations. Additionally, the prompt can include the path to an image annotated in red (with arrows, text, areas, etc.) (see Figure 3). The second input is a success function, which must be a user-defined function tailored to their environment tailored to their environment. This function determines how success is manifested within the specific context of the task. Finally, users need to provide objective metrics. These metrics serve to enhance the feedback provided to the LLM during the refinement phase.

2.2 Initial Generation

We have chosen to implement a zero-shot user prompt. Although studies have demonstrated the effectiveness of few-shot prompting [1], we believe that a prompt without examples offers users greater simplicity and scalability in any environment. The image will be charged to the LVLM and used as a reference to better capture the information about the described environment and the objective to be achieved. Indeed, the LVLM has a better design for the reward function that aligns with the goal in this case.

Among the strategies employed to enhance zero-shot generation, one particularly effective method is step-back prompting [16]. (see Figure 1.2). Thus, we have implemented a collaboration between two instances of LLM (Ollama Chat), where one acts as the critic providing additional context to the specialized code actor, who will generate the reward function. This methodology has led us to improvements in the generated function (see Figure 1.3).

To avoid potential errors, the generated code is automatically checked for syntax errors and logical issues (see Figure 1.4). If any errors are found, they are sent back to the model via a secondary prompt, allowing it to revise its output. This process helps the model improve over time and reduces the chances of producing invalid outputs.

2.3 Learning a Policy

The generated reward function takes the observations as parameters, along with two boolean values indicating whether there is a success and whether there is a failure. Once the initial reward function is generated, we have the agent search for its policy using the specified learning algorithm to *stable baseline 3* [10]. We use two algorithms for our tests: Deep Q Network [9](DQN) and Proximal Policy Optimization [11](PPO). PPO is a highly popular algorithm due to its versatility across a wide range of environments, particularly those with continuous observations or actions, as well as its ease of parameter tuning and efficiency.

During training (see Figure 1.5), rewards and observations are retrieved, and if the user wishes to, they can implement objective metric functions that take these observations and return a dictionary of useful objective metrics for this environment. During testing, we evaluate based on the Success Rate defined by the user and specific to each environment. These objectives serve two purposes: to compare the performance of the generated reward function with the baseline "legacy" reward function and to compare multiple custom reward functions when they are generated in parallel.

2.4 Refined Reward

The refinement process unfolds as follows:

1. **Human feedback: (see Figure 1.6**) The training of the agent is visualized in a video, allowing a human observer to analyze the agent's behavior. The observer provides annotations or comments on desirable or undesirable behaviors. These insights are integrated into the system to refine the reward function, ensuring better alignment with user-defined goals.
2. **Video observation by an LVLM: (see Figure 1.6**) As an alternative or complement to human feedback, VIRAL enables an LVLM to directly observe the training video. The LVLM used is LLAVA-VIDEO, and its role is to analyze the visual sequence to identify problematic or misaligned behaviors. The generated response will focus on the agent's movement and will describe the agent's positions in space. These detailed observations are used to enhance the reward function without human intervention, allowing for a more autonomous process to address any alignment issues.
3. **Automated critique and improvement: (see Figure 1.7**) The critic LLM analyzes the training results by leveraging collected statistics. This LLM identifies potential reasons why the reward function underperformed. These observations are passed to the actor LLM, which creates an improved reward function, addressing the identified weaknesses. The goal is to propose a function that enhances agent performance and aligns more closely with the intended objectives.

These methods are combined into an iterative approach: in each cycle, the refined reward function is evaluated and compared to the previous version to ensure it provides significant improvement. If it does not, the refinement process is repeated until the desired performance level is achieved.

By combining artificial intelligence with vision capabilities, VIRAL delivers optimized reward functions. This iterative and modular process provides flexibility and adaptability, making it suitable for a wide range of reinforcement learning environments and surpassing traditional reward function design methods.

3 Experiments and Results

During our work, we chose to use *Qwen2.5-Coder* as the actor due to its performance being comparable to that of *GPT-4*, making it a reliable choice for this role. For the critic, we chose *Llama3.2-Vision* [8], as it is lightweight, open-source, and well-suited for our framework's requirements. Nevertheless, our framework is flexible and allows the use of any LLM available through Ollama. For video description tasks, we implemented a dedicated server that runs *llava-video-7b-qwen2* [15].

In our experiments, we used a selection of environments from the Gymnasium toolkit to evaluate the performance of our generated reward functions. We started our work with the classic environments of **Cart Pole** and **Lunar Lander**, which allowed us to test fundamental control and optimization strategies in simple yet challenging scenarios. Next, we explored the **Highway** environment, where a vehicle must navigate a multi-lane road, avoiding collisions and optimizing its speed while adhering to driving rules. Given the increasing relevance of autonomous vehicles, we considered it essential to include this environment in our study as it addresses modern-day challenges in automation and decision-making. Finally, we incorporated two robotics-focused environments, **Hopper** and **Swimmer**, both derived from the MuJoCo physics simulator. These environments played a

crucial role in evaluating the efficiency of our reward function in comparison to Gymnasium's, as they allowed us to investigate intricate robotic locomotion and control systems.

All our project results are available in CSV format in our GitHub repository.

3.1 Comparison of efficiency against Gymnasium reward functions

In this section, we evaluate the efficiency of our method by comparing the learning curves obtained using our generated reward function with those derived from the baseline (legacy) reward function. These comparisons are conducted in the CartPole environment, which is particularly well-suited for analyzing learning speed and stability.

We compare in Figure 2 rolling means over 10 runs of learning to demonstrate that our approach enables the agent to learn faster and achieve superior performance compared to using the baseline reward function. The results show a quicker convergence of the normalized cumulative reward, whereas the gymnasium baseline reward function remains significantly less effective throughout training. Through Table 1, we observe that the policy learned with our reward function outperforms that of Gymnasium. Additionally, we can see that the cart moves slightly more on average, resulting in a more stable pole overall.

These rewards functions were discovered using PPO and qwen2.5-coder:32b as an actor/critic, with this simple goal prompt: *"Create a reward function for the CartPole environment that encourages keeping the pole upright for as long as possible."*

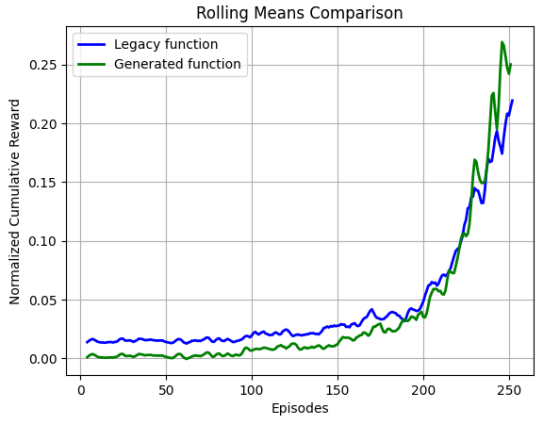

Figure 2. 10 runs of PPO with our reward function and with the reward function of Gymnasium

Table 1. Comparison between legacy reward and ours (over 10 runs)

reward function	cart position diff	pole angle diff	success rate
gymnasium	0.281202	0.064587	0.58700
ours	0.308173	**0.062499**	**0.85300**

For the fast highway environment using DQN, we have better success rates than the basic function **7 times out of 10**, using only qwen2.5-coder in version 7b for the critic and actor. On these 10 runs, we obtain **a median success rate of 0.78**. The goal prompt was: *"Control the ego vehicle to reach a high speed without collision."*

These results highlight the capability of our method to generate tailored and efficient reward functions, significantly reducing the training time needed to achieve complex objectives.

3.2 Aligned

By including an annotated image with the very first prompt, the learned policy performs as the user desires. This is evident with the swimmer example, where we provide an annotated Figure3 and achieve the desired policy Figure4.

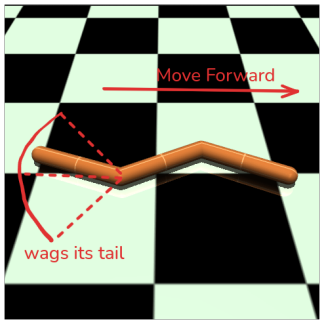

Figure 3. Annotated image given to Llama3.2-vision

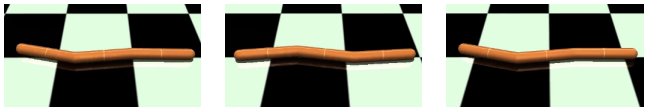

Figure 4. The policy discovered thanks to annotated images.

3.3 Behaviors

For the LunarLander environment, we successfully trained the agent to perform a novel behavior: stationary flight. This was achieved using a single annotated image and the following prompt: *"Do not crash but do not land; I want to make a stationary flight."*

4 Conclusion

In this study, we demonstrated that our approach outperformed legacy reward functions in terms of performance and flexibility. Our results showed that agents were able to learn new behaviors based on simplistic sketches, highlighting the robustness of our learning methodology. Furthermore, the integration of video feedback or human feedback into our experiments enabled agents to better align with the intended objectives, providing a deeper understanding of the desired behaviors.

Looking forward, an intriguing avenue to explore would be the adaptation of pre-existing policies to learn new behaviors. This approach could pave the way for greater policy generalization and smoother transitions between different sets of complex tasks.

5 Citations and references

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

[3] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[4] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

[5] B. Lin, Y. Ye, B. Zhu, J. Cui, M. Ning, P. Jin, and L. Yuan. Video-llava: Learning united visual representation by alignment before projection. *arXiv preprint arXiv:2311.10122*, 2023.

[6] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[7] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[8] Meta. Llama-3.2-11b-vision-instruct. https://huggingface.co/meta-llama/Llama-3.2-11B-Vision-Instruct, 2024.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning, Dec. 2013.

[10] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017.

[12] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma. Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics. *arXiv preprint arXiv:2309.06687*, 2023.

[13] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[14] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[15] Y. Zhang, J. Wu, W. Li, B. Li, Z. Ma, Z. Liu, and C. Li. Video instruction tuning with synthetic data. *arXiv preprint arXiv:2410.02713*, 2024.

[16] H. S. Zheng, S. Mishra, X. Chen, H.-T. Cheng, E. H. Chi, Q. V. Le, and D. Zhou. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*, 2023.