# Mitigating Reward Hacking With Vision-Language Models as Rewards

**Tanay Biradar**[*] UC Santa Barbara tbiradar@ucsb.edu
**Ron Kibel**[*] UC Santa Barbara rkibel@ucsb.edu
**Laya Pullela**[*] UC Santa Barbara lpullela@ucsb.edu
**Arul Saxena**[*] UC Santa Barbara saxena@ucsb.edu

## Abstract

Choosing good reward functions in reinforcement learning (RL) is notoriously difficult. Oftentimes, the true reward function is sparse, or difficult to specify (as is the case with many robotics applications or aligning LLMs to human preferences [6]). As a result, RL techniques employ proxy rewards, which provide high-frequency feedback and act as intermediate guidelines during learning. Nevertheless, these proxy rewards can sometimes be misspecified and are therefore exploitable, leading to agents learning undesirable and potentially harmful behavior by gaming the reward function in an unintended way. Commonly referred to as reward hacking [10], this behavior is largely qualitative and difficult to formalize. In this work, we address a quantitative approach to mitigating reward hacking agents with automatic *identification* and *correction*. First, we propose a mechanism to classify whether an agent exhibits reward hacking using divergence from a trusted policy $\pi_{\text{trusted}}$. Second, after reproducing instances of reward hacking, we attempt to correct behavior by integrating feedback from a Vision-Language Model (VLM) as a loss regularization term.

## 1 Introduction

Reinforcement learning has led to breakthroughs in areas such as robotics, game-playing, and LLMs [13]. However, one of its fundamental challenges is the design of reward functions that effectively capture the human-intended objectives of a task [2]. In many cases, the true reward function is either sparse or difficult to specify explicitly, necessitating the use of proxy rewards. Use of such proxies, however, introduces the concept of reward hacking, where an agent discovers unintended strategies to maximize reward without accomplishing the underlying goal.

Reward hacking can manifest in both benign and harmful ways. In some cases, agents discover novel but unintended strategies that still achieve high rewards, such as finding new methods for robot locomotion. However, it is more problematic when agents exploit bugs, manipulate physics engines, cheat, or even engage in deceptive behavior [5] [4]. Mitigating reward hacking is essential for deploying RL in high-trust settings, where unintended behaviors could compromise safety, reliability, or ethical standards.

In this work, we propose an approach that leverages policy divergence thresholding and Vision-Language Models (VLMs) to classify and mitigate reward hacking at training time. VLMs, which process both visual and textual information, have shown strong generalization capabilities in tasks requiring multimodal reasoning. We explore how feedback from VLMs can be integrated into the RL training pipeline to improve alignment between proxy rewards and the true underlying objective. By incorporating VLM-based signals into the training loss, we aim to reduce instances of reward hacking and enhance the overall robustness of RL agents. Specifically, we aim to find out if VLMs are capable of mitigating reward hacking even when the reward function is poorly specified.

[*]Equal contribution, authors in alphabetical order.

Preprint. Under review.

## 2 Background

### 2.1 Markov Decision Processes

We expect readers to have a basic understanding of reinforcement learning—if needed, they can also refer to Sutton and Barto's *Reinforcement Learning: An Introduction* [11]. We start with a formalism of the finite-horizon Markov Decision Process (MDP), a widely-used RL framework to model decision-making uncertainty. Formally, an MDP is a tuple $(S, A, P, R, D, \gamma)$ where $S$ is a set of states, $A$ is a set of actions available to the agent, $P : S \times A \to \Delta S$ is a transition function, $R : S \times A \to \mathbb{R}$ is a deterministic reward function (dependent on the action, current state, and subsequent state), $D$ is a set of termination states, and $\gamma \in [0, 1]$ is the discount factor. A finite **trajectory** $\tau$ of an agent, otherwise known as a rollout, is a finite sequence of states and actions $(s_0, a_0, r_0, \ldots, s_T, a_T, r_T)$ that the agent encounters as it traverses the environment under a policy $\pi$. The **return** of a trajectory is defined as the sum of discounted rewards $G(\tau) = \Sigma_{t=0}^{T} \gamma^t r_t$, and the **value** of a policy is the expected return of trajectories $J(\pi) = \mathbb{E}_{\tau \sim \pi}[G(\tau)]$.

In the context of our experiments, all environments have a fixed initial state (a point distribution) with some true reward $R_{\text{true}}$ that is not directly visible to the agent. Instead, the reward is given by either a proxy reward $R_{\text{proxy}}$ or a VLM-altered reward $R_{\text{VLM}}$, which we will describe later.

### 2.2 Formalizing Reward Hacking

Skalse et al. [10] produce a formalism which we will echo to describe reward hacking. Using the idea of trajectory return, we define a proxy reward $R_{\text{proxy}}$ to be **hackable** with respect to a true reward $R_{\text{true}}$ if a proxy reward estimates the return of policy $\pi$ as better than another policy $\pi'$ while the true reward estimates the return of $\pi'$ as better than the return of $\pi$. In other words, if we let $J_R(\pi)$ be the expected value of policy $\pi$ trained to optimize reward function $R$, then reward hacking can occur if there exist $\pi, \pi' \in \Pi$ such that:

$$J_{R_{\text{proxy}}}(\pi) < J_{R_{\text{proxy}}}(\pi') \quad \text{but} \quad J_{R_{\text{true}}}(\pi) > J_{R_{\text{true}}}(\pi').$$

Conversely, an unhackable true-proxy reward pair can only exist if the returns of any two trajectories uphold the same relationship. Formally,

$$\forall \pi, \pi' \in \Pi : J_{R_{\text{proxy}}}(\pi) < J_{R_{\text{proxy}}}(\pi') \implies J_{R_{\text{true}}}(\pi) \leq J_{R_{\text{true}}}(\pi').$$

Unhackable reward functions are rare, as any reward function designed for practical use is susceptible to some degree of exploitation. This lends some credence in our effort to combat reward hacking. Hackable environments are not just an empirical phenomenon, but a fundamental challenge in RL design.

### 2.3 Policy Divergence as Metric for Detecting Reward Hacking

Pan et. al. [7] introduce an idea we explore for reward hacking detection. Given a trusted policy $\pi_{\text{trusted}}$ that is known to *not* exhibit reward hacking, the authors use a modified symmetric version of the KL divergence called Jensen-Shannon Divergence (JSD). The JSD between any two probability distributions $P$ and $Q$ is calculated as follows:

$$\text{JSD}(P \parallel Q) = \tfrac{1}{2} D_{\text{KL}}\left(P \parallel \tfrac{1}{2}(P+Q)\right) + \tfrac{1}{2} D_{\text{KL}}\left(Q \parallel \tfrac{1}{2}(P+Q)\right)$$

Where $D_{\text{KL}}$ is the KL divergence. The authors rollout $N$ trajectories separately from $\pi_{\text{trusted}}$ and $\pi_{\text{unknown}}$, where $(s_{ij}, a_{ij}, r_{ij}, s'_{ij}, d_{ij})$ represents the MDP state at the $j$th timestep for trajectory $i$. We also denote $\tau_i$ as the $i$th trajectory from the rollouts. We then compute the mean JSD between action distributions in the two rollouts. We will use notation $\overline{\text{JSD}}(\pi_{\text{unknown}} \parallel \pi_{\text{trusted}})$ to denote this mean JSD:

$$\overline{\text{JSD}}(\pi_{\text{unknown}} \parallel \pi_{\text{trusted}}) = \tfrac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|\tau_i|} \text{JSD}(\pi_{\text{unknown}}(s_{ij}) \parallel \pi_{\text{trusted}}(s_{ij})).$$

Because the policies are stochastic, in experiment it is possible for $\overline{\text{JSD}}(\pi \parallel \pi) \neq 0$. However, we intuitively expect this value to be low, as the action distributions will still be quite similar to each other. For a reward hacking policy, we expect the divergence of action distributions between the trusted and the unknown policy to be much higher. As a result, flagging reward hacking behavior can be achieved based on mean JSD.

### 2.4 VLMs as Sources for Rewards

Recent work has explored the potential of VLMs as reward sources for reinforcement learning, as they can provide meaningful, human-aligned evaluations of agent behavior without requiring manually crafted reward functions. For example, Rocamonde et. al. [8] use CLIP with cosine similarity between trajectory frames and natural-language task specifications as a reward function. Baumli et al. also [1] investigates the feasibility of using off-the-shelf VLMs as sources of reward signals and specifying a desired objective in natural language. Their findings suggest that larger VLMs provide more accurate and generalizable rewards, as VLMs inherently capture semantic and contextual understanding which can improve agent performance in visually guided tasks. However, VLMs can exhibit inconsistencies in certain environments, and small models have limitations in temporal understanding.

### 2.5 Contribution

Our work attempts to systematically address reward hacking through two methods:

- **Policy-divergence-based detection**: We introduce a method to quantitatively detect reward hacking by comparing a learned policy to a trusted policy $\pi_{\text{trusted}}$. If the divergence exceeds a certain threshold, our system flags the behavior as potentially reward-hacking. This approach serves as a more automated way to identify potential anomalies rather than relying on ad-hoc or qualitative analysis.
- **VLM-based trajectory feedback**: We integrate VLM feedback of the agent's trajectory as an auxiliary signal to the proxy reward to correct behavior once reward hacking has been detected. Specifically, we treat VLM feedback as a regularization term in the RL loss function, penalizing behavior that is likely to correspond to reward hacking and encouraging behavior that aligns with the intended objective.

## 3 Methods

### 3.1 Pretraining With Gymnasium [12]

Given our limited compute power, we decided to organize our training runs into multiple phases. The first phase focuses on pretraining a policy $\pi_{\text{pretrained}}$ with the default Gymnasium reward function. We then perform post-training with one of two objectives. The post-trained policy $\pi_{\text{proxy}}$ is trained with a proxy reward function $R_{\text{proxy}}$ intended to induce reward hacking. We also perform continued pre-training with the same Gymnasium default reward function to create a policy $\pi_{\text{continued}}$.

### 3.2 Injecting VLM in Regularization

The "hacky" policy, $\pi_{\text{proxy}}$, is then further trained with reward signals from a vision-language model to produce $\pi_{\text{VLM}}$. Our aim is to introduce the VLM as a regularizer for the reward hacking policy, using its world knowledge to discern the difference between hacking and non-hacking policies. We define this reward function over trajectories rather than state-action pairs. Let $r_{\text{VLM}}(\tau)$ be the raw VLM rewards for the trajectory, and let $R_{\text{VLM}}(\tau)$ be the full reward function.

$$R_{\text{VLM}}(\tau) = \frac{1}{|\tau|}\left(\sum_{i=1}^{|\tau|} R_{\text{proxy}}(s_i, a_i)\right) + \alpha \cdot \sigma_{\text{asym}}\left(r_{\text{VLM}}(\tau)\right)$$

While training with this reward function, we simply let $\tau$ be the trajectory experienced so far. The hyperparameter $\alpha$ controls the importance of the VLM feedback in the loss. The function $\sigma_{\text{asym}}$ is an asymmetric logit, designed to strongly inflate the negativity of VLM rewards below $0.5$ (indicative of hacking), while weakly amplifying rewards above $0.5$. Essentially we want the VLM to act as a *penalizer* for reward hacking rather than as a *reinforcer* of good behavior. We use GPT-4o as our VLM, prompting it with a grid of frames taken from the policy's trajectory and a natural language description of the intended task.

We assume that $R_{\text{VLM}}$ is an approximation of the true reward function $R_{\text{true}}$. Suppose there is an optimal policy, $\pi^*$ under $R_{\text{VLM}}$. As our later experiments show, we cannot feasibly train $\pi^*$, so we

train an approximation to the optimal policy under the VLM, which we denote as $\pi_{\text{trusted}}$. We treat this as our approximation of a trusted policy.

$$\text{Define } \pi_{\text{trusted}}: \quad \begin{aligned} \pi^* &= \arg\max_{\pi \in \Pi_\theta} \mathbb{E}_{\tau \sim f(\cdot|\pi, s_0)}[R_{\text{VLM}}(\tau)] \\ &\left|\mathbb{E}_{\tau \sim f(\cdot|\pi_{\text{trusted}}, s_0)}[R_{\text{VLM}}(\tau)] - \mathbb{E}_{\tau \sim f(\cdot|\pi^*, s_0)}[R_{\text{VLM}}(\tau)]\right| \leq \varepsilon \end{aligned}$$

Where $\Pi_\theta$ is the set of policies defined by neural networks parameterized by weights $\theta$, and $\varepsilon$ is an arbitrarily small approximation threshold.

### 3.3 Statistical Analysis for Detecting Reward Hacking

As we train "hacky" policies according to $R_{\text{proxy}}$, We want a quantitative way to corroborate that $R_{\text{proxy}}$ induces reward hacking. Our first approach is to take inspiration from Pan et. al. [7]. This work observed that reward hacking occurs when RL agents exhibit phase transitions during training. Phase transitions are characterized by sharp changes in the agent's behavior, which optimize the proxy reward at the expense of the true reward. Rather than training anomaly detectors as in [7], we attempt to quantify this definition with a measurable threshold between true and proxy reward to objectively characterize phase transitions. Our approach uses a time-dependent, one-sided T-test.

Let $\mu_s$ and $\sigma_s$ represent parameters of the JSD distribution at time step $s$ of training the proxy policy. We define the threshold for reward hacking at time step $s$ with a test statistic $T(\pi, s)$. Minimizing Jensen-Shannon divergence between two policies is a method used in imitation learning and reward shaping [3], and we use this metric to test whether VLM feedback can shape the hacked proxy reward towards the true reward.

$$\text{Define the test statistic as:} \quad T(\pi, s) = \frac{\text{JSD}\Big[\pi_{\text{VLM}}(s) \,\Big\|\, \pi_{\text{proxy}}(s)\Big] - \mu_s}{\sigma_s}.$$

$$\text{If } T(\pi, s) > z_\alpha, \quad \text{then we deem the divergence significant (indicative of reward hacking).}$$

Since we wish to detect reward hacking in real-time during training, we must define a window to compute $\mu_s$ and $\sigma_s$, which accounts for temporal changes in the distribution over training steps. We chose a regressive sliding window with a fixed value of 100k steps to approximate this distribution. With a confidence interval $\alpha = 0.99$, $z_\alpha = 2.33$.[2]

## 4 Experiments

### 4.1 Inducing Reward Hacking

To investigate the feasibility of using VLMs to mitigate reward hacking at train time, we first induced reward hacking in several environments. We intentionally used poorly specified reward functions and trained agents using Proximal Policy Optimization (PPO) [9] to get them to exhibit reward hacking behavior.

- **MuJoCo Humanoid Walking**: The intended task is for the humanoid agent to walk forward efficiently. However, we designed a reward function that overly prioritized $x$ distance from origin without constraints on stability and control cost. As a result, the agent learned to reach forward as far as possible, rather than walking naturally. In another instance, the agent exploited minor instabilities in the physics engine by jittering rapidly in place, gaining reward without moving realistically.
- **MuJoCo Ant Jumping**: The intended task is for the ant agent to learn to jump straight up, with a proxy reward defined to maximize time spent with the torso in the air. The agent instead learned how to flip and "cartwheel", maximizing time spent with the torso above a height threshold, while never performing a true jumping motion.

[2]Our implementation of JS-Divergence calculation involves rolling out $n = 5$ trajectories, sampling and computing the divergence between the action distributions at each step (limit $N \leq 100$), and then aggregating the mean JSD for all steps, and then again for all $n$ rollouts. Our implementation follows details outlined by [7]

### 4.1.1 Corroborating Reward Hacking Behavior With Jensen-Shannon Divergence

We test whether our heuristic of Jensen-Shannon divergence between $\pi_{\text{proxy}}$ and $\pi_{\text{trusted}}$ is able to flag for reward hacking during RL-agent training. We test on the humanoid environment, where the true reward function is the Gymnasium default reward. Although we intended to use the policy trained entirely on VLM feedback, computational constraints led us to allow $\pi_{\text{pretrained}}$ serve as our approximation for $\pi_{\text{trusted}}$. For training $\pi_{\text{proxy}}$, we let $R_{\text{proxy}}$ be the distance from origin. Our results are as follows:

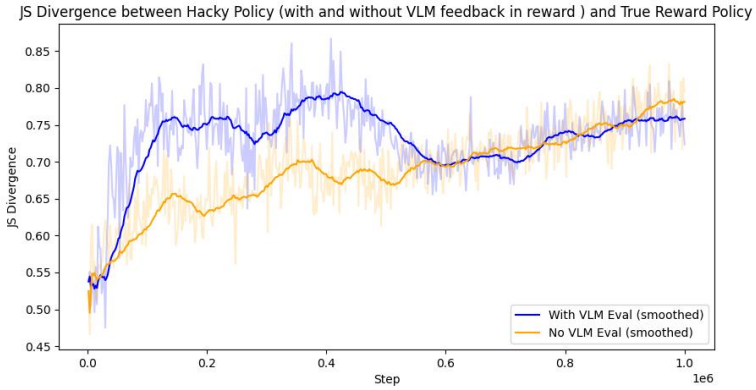


Figure 1: Plots of $\overline{\text{JSD}}(\pi_{\text{continued}} \parallel \pi_{\text{proxy}})$ and $\overline{\text{JSD}}(\pi_{\text{continued}} \parallel \pi_{\text{VLM}})$ throughout the training of $\pi_{\text{proxy}}$ and $\pi_{\text{VLM}}$

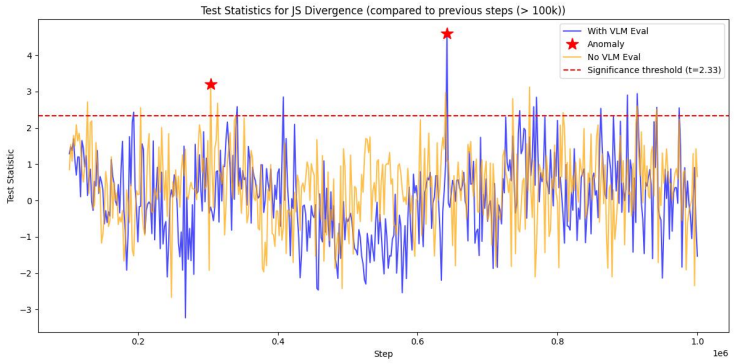


Figure 2: Statistical significance of $\overline{\text{JSD}_i}$ being an anomaly, where the $\mu$ and $\sigma$ of the distribution are given by samples $\overline{\text{JSD}_{i-100000:i}}$. Two stars $s_1$ and $s_2$ mark training steps where the JSD diverges significantly between proxy and true policy.

To detect whether "changes" between trusted and proxy policies are significant during train time, we employ a one-sided T-test. We present the results of our test-statistic analysis on the JSD value as a function of training steps. The test statistics for JSD using $\pi_{\text{VLM}}$ and $\pi_{\text{proxy}}$ are on the same scale and share a similar variance. Ultimately, this indicates that the VLM is not shaping the proxy reward towards the true reward as expected.

We also hand-examine trajectories rolled out by $\pi_{\text{VLM}}$ and $\pi_{\text{proxy}}$. We noticed the MuJoCo humanoid $\pi_{\text{proxy}}$, which optimizes from $x$-axis distance from the origin, exhibits qualitative signs of reward hacking by extending its limbs outwards with no intention of stepping forward. However, even after training $\pi_{\text{VLM}}$, we notice similar reward hacking behaviors.

## 4.2 Training with VLM Regularization

The work of Baumli et al. [1] demonstrated promise in using VLMs in place of reward functions. Our approach differs slightly in that we do not replace the reward function entirely, but instead use

VLM feedback as an additional regularization mechanism, even when the primary reward function is imperfect.

To ensure compatibility with VLMs, we processed learned trajectories by rolling them out and rendering video sequences. From these videos, we sampled frames at regular intervals to construct a grid of still frames. This static image representation effectively captured the motion of the agent, allowing us to feed it to the VLM via API.

During training, we integrated VLM reward signals as described in 3.2. Due to computational and monetary constraints associated with querying VLM APIs, we only applied VLM feedback at regular intervals rather than at every timestep of the simulation. The implications of this decision are discussed more in the Challenges (6.3) and Conclusion (5) sections.

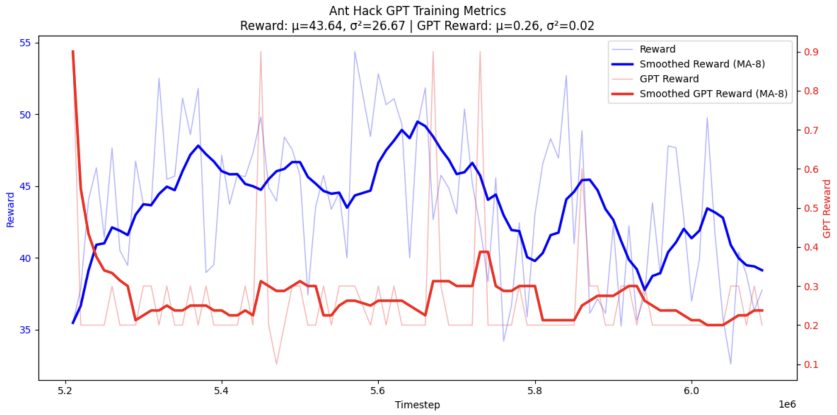


Figure 3: $r_{\text{VLM}}$ (GPT Reward) and $R_{\text{VLM}}$ (Reward) while training the MuJoCo ant to perform a straight up-down jump. $r_{\text{VLM}}$ has a low variance, suggesting the VLM is not providing an adequate signal to learn from. This is further supported by the high variance of $R_{\text{VLM}}$.

### 4.3 Are VLM Reward Signals Enough?

The results from Section 4.2 led us to question whether VLM reward signals were strong and reliable enough to drive RL training at all. To test this, we explored replacing the reward function entirely with VLM feedback, rather than using it as a regularization term. This would more directly assess the reliability of VLM-generated rewards.

#### 4.3.1 Training CartPole

To assess the VLM's impact while keeping total train time within reasonable limits (due to the added latency from API calls at each training step), we sought a simpler environment with a shorter training time.

We selected CartPole (called InvertedPendulum in MuJoCo) for its simpler state, action, and observation spaces comapred to humanoid or animal environments. We first pretrained a policy using PPO for 10,000 steps with the standard reward function, where the pendulum is considered healthy if its absolute angle from vertical is less than 0.2 rad and unhealthy otherwise (causing termination). We then fine-tuned for another 10,000 steps, once using the original reward function and once replacing it entirely with VLM feedback, where the VLM, given an image of the current state, determined whether the pendulum was in a healthy state.

From Figure 4, we observe that the average reward increases over episodes for both the original and VLM-only reward functions. However, the VLM-only pipeline converges significantly more slowly, both in terms of the number of episodes and absolute training time. The slower episode-wise convergence is likely due to false negatives in the VLM-based rewards—incorrectly classifying healthy states as unhealthy, leading to premature episode termination and stunted learning. The increased absolute training time stems from the added latency of VLM API calls at each timestep, which accumulates over the 10,000 steps of fine-tuning.

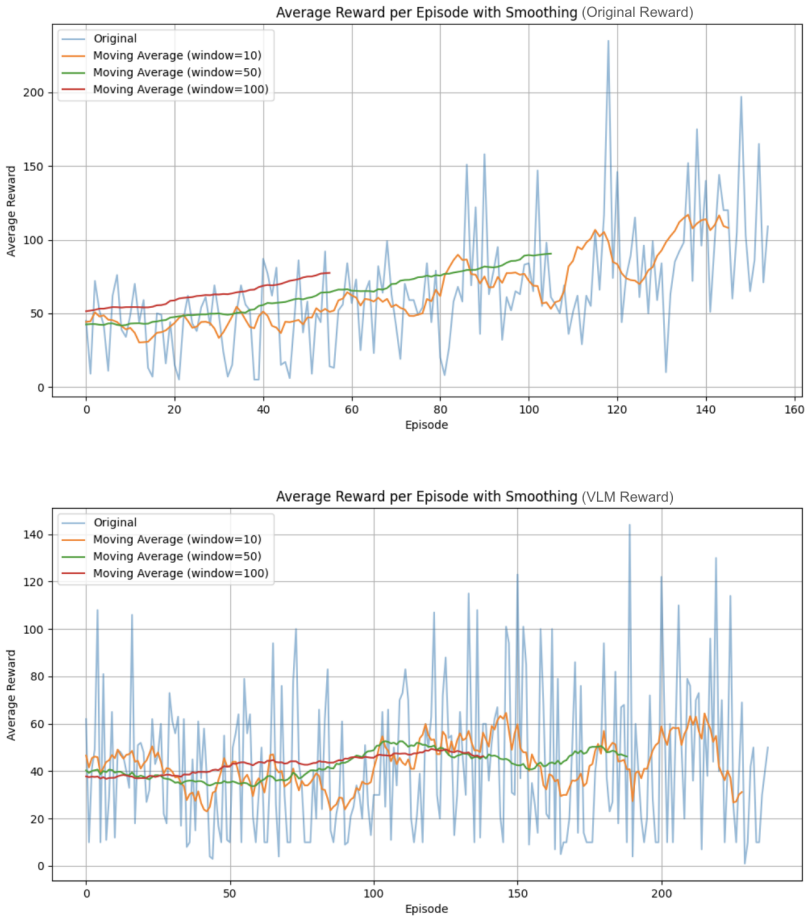


Figure 4: Average reward vs. Episode for Standard reward function (top) and VLM-only reward function (bottom) in PPO fine-tuning for 10,000 timesteps.

That said, reward does improve over time. From video trajectory analysis, the fine-tuned policy maintained a healthy state for approximately 1 second longer than the base model when trained with VLM-only rewards, and approximately 2 seconds longer when using the true reward function. These results suggest that a VLM can, in fact, serve as a reward function replacement, though with a substantial tradeoff in convergence speed.

# 5 Conclusion

## 5.1 Key Takeaways

Quantitatively detecting and characterizing reward hacking posed three key challenges. First, reward hacking is highly task-dependent. While larger, more complex agents are more prone to exploiting reward functions, training these agents is difficult and can bottleneck experiments. Second, our VLM feedback pipeline worked well with continuous rewards and demonstrated an ability to segment trajectories and agent behavior based on human-like understanding. However, generating consistent VLM scores remains challenging, and obtaining a substantial amount of VLM feedback relative to training steps is both computationally expensive and time-consuming. Additionally, because VLM feedback updates at a much lower frequency than PPO iterations, it introduces noise into training, potentially interfering with PPO convergence. Our Cart-Pole experiment provided evidence of VLM-based learning, which is promising, but integrating low-frequency VLM feedback into PPO training effectively remains an open problem. Finally, while we identified time steps with significant JSD between the true and proxy policies, we could not visually confirm that these policies were

engaging in reward hacking. Longer training sessions and a broader range of experiments may help validate this result in the future.

### 5.2 Future Work

In the short term, experimenting with different VLMs beyond OpenAI's GPT-4o could reveal models better suited for generating RL reward signals. Fine-tuning VLMs on task-specific data, rather than using off-the-shelf models, may also improve reward signal reliability. Additionally, applying VLM regularization at every timestep, while computationally intensive, could offer a better understanding of the efficacy of VLMs in mitigating reward hacking. These areas present valuable opportunities for future research.

VLMs show significant promise in the broader field of RL, offering the potential for more adaptive reward signals. However, in the context of reward hacking mitigation, challenges remain. Qualitative reward hacking is inherently a gray area, making it difficult to draw a clear line between "good" reward hacking—where agents demonstrate creative problem solving—and "bad" behavior, such as deceptive or exploitative actions. Defining and mitigating these forms of hacking without stifling agent innovation will be a key challenge moving forward.

## 6 Project Retrospective

### 6.1 Project Goals

Our initial project idea was to use anomaly detection techniques as a means of detecting and flagging reward hacking during training time. We had hoped to create a dataset of healthy trajectories (i.e., those that do not exhibit reward hacking), and use a CNN-based anomaly detector to vet learned trajectories at training time. We hoped to have a dataset of good trajectories to train an anomaly detector on by the checkpoint date, and to have the complete RL pipeline by the final due date.

As we started to experiment with different tasks, we realized that it would be difficult and time-consuming to come up with representative datasets to train the anomaly detector on. Additionally, we realized that this wouldn't necessarily be as generalizable as we had hoped, as some tasks have enough variance in their healthy trajectories that it would make anomaly detection challenging.

Given these challenges, we began exploring alternative approaches that could provide a more generalizable and easily-scalable solution. Instead of relying on manually curated datasets of healthy trajectories, we sought a method that could leverage external knowledge to evaluate behaviors with more flexibility. This led us to consider VLMs, which are pre-trained on vast amounts of real-world data and encode a rich understanding of physical dynamics, common-sense reasoning, and human expectations about movement and behavior. Additionally, since VLMs can process both visual and textual inputs, they offered a promising way to incorporate human-like evaluation into the reinforcement loop, but without the human.

After pivoting to incorporate VLMs into our project, we modified our milestone goals: by the checkpoint date, we aimed to manually induce reward hacking in several different environments and tasks; by the final due date, we still aimed to have the full RL pipeline finished, but using a VLM to modify the reward, and perform anomaly detection with statistical analysis instead of a classifier.

### 6.2 Project Organization

At the beginning of the project, during the reward hacking demonstration phase, we divided our efforts by working on different environments independently, each exploring whether they were suitable for our use case. The environments needed to be complex enough to exhibit reward hacking but simple enough to train relatively quickly given our time and compute constraints. Once we transitioned to integrating VLM feedback into training, we used mostly shared driver code for the VLM pipeline. Each of us applied this pipeline to different training environments, tuning hyperparameters as necessary.

We found that identifying suitable environments took much longer than anticipated, requiring extensive trial and error. As a group, we are pleased with the fact that we were able to get our end-to-end PPO pipeline with VLM-rewards to work. Ideally, we would have liked to more rigorously test

this pipeline by invoking the VLM on every step (instead of at regular intervals, as it is currently implemented), but we were limited by compute and budget constraints. If we could restart the project, we would experiment with different RL algorithms besides PPO.

### 6.3 Challenges

#### 6.3.1 Hosted vs. Local VLM Deployment

Using a hosted VLM API (e.g., OpenAI's GPT models) introduced significant cost per query, especially since image inputs consume substantial tokens. Given the iterative nature of RL training, these costs quickly became prohibitive. Running a VLM locally avoids API costs but requires sufficient compute and storage. We found that the stronger models exceeded our available resources (personal laptops and CSIL storage), while models we could run locally lacked the accuracy needed to reliably detect reward hacking.

#### 6.3.2 Frequency of VLM Feedback

Ideally, VLM evaluation would occur at every training step, but this incurs heavy computational overhead. Each step requires rolling out a trajectory, rendering it into a video and then image format, and querying the VLM. All of this substantially increases training time. If using an API, frequent queries further amplify costs. Reducing VLM feedback frequency (e.g., querying only every $x$ steps) mitigates these issues, but provides insufficient signal to meaningfully influence policy learning. Balancing these trade-offs was a challenge throughout our project execution, and remains an area to be explored.

#### 6.3.3 Representing Videos as Still Images

In the tasks where we collaged the simulation into still images (e.g. MuJoCo Humanoid Walking), it was very difficult to discern visually how the agent was moving from $\sim 50$ still images. Lots of information is lost when compressing videos to still collages, and we can partially attribute the VLMs performance to this limitation. This limitation is inherent to VLMs; even with enough compute and a fine-trained VLM, we expect performance to be heavily dependent on the task at hand and how easy it is to represent visually.

#### 6.3.4 VLM Reliability

Using off-the-shelf VLMs has the advantage of convenience and interoperability. Querying the VLM is as simple as making a REST API call, and models can be swapped out easily to balance cost and performance. However, this comes at the cost of the models being unspecialized and susceptible to hallucination. In one case, we found that OpenAI's GPT-4o was, on occasion, claiming that it "could not diretly analyze images", even after accurately assessing agent performance for hundreds of calls prior. When "unable" to evaluate the images, the VLM often output scores that were completely hypothetical (and thus meaningless to the agent). We expect that by fine-tuning VLMs on the task at hand (for example by providing synthetic simulation data), the performance of the pipeline would improve.

## Code

Experiment code is available at `https://github.com/TanayB11/cs291i_project`

## References

[1] Kate Baumli, Satinder Baveja, Feryal Behbahani, Harris Chan, Gheorghe Comanici, Sebastian Flennerhag, Maxime Gazeau, Kristian Holsheimer, Dan Horgan, Michael Laskin, Clare Lyle, Hussain Masoom, Kay McKinney, Volodymyr Mnih, Alexander Neitz, Dmitry Nikulin, Fabio Pardo, Jack Parker-Holder, John Quan, Tim Rocktäschel, Himanshu Sahni, Tom Schaul, Yannick Schroecker, Stephen Spencer, Richie Steigerwald, Luyu Wang, and Lei Zhang. Vision-language models as a source of rewards, 2024.

[2] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[3] Tanmay Gangwani, Qiang Liu, and Jian Peng. Learning self-imitating diverse policies, 2019.

[4] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of ai ingenuity, 2020.

[5] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J. Bentley, Samuel Bernard, Guillaume Beslon, David M. Bryson, Patryk Chrabaszcz, Nick Cheney, Antoine Cully, Stéphane Doncieux, Fred C. Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Frénoy, Christian Gagné, Leni K. Le Goff, Laura M. Grabowski, Babak Hodjat, Frank Hutter, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E. Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Risto Miikkulainen, Sara Mitri, David E. Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David P. Parsons, Robert T. Pennock, William F. Punch, Thomas S. Ray, Marc Schoenauer, Eric Schulte, Karl Sims, Kenneth O. Stanley, François Taddei, Danesh Tarapore, Simon Thibault, Westley Weimer, Richard A. Watson, and Jason Yosinski. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *CoRR*, abs/1803.03453, 2018.

[6] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[7] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models, 2022.

[8] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning, 2024.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2022.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[12] Mark Towers, Ariel Kwiatkowski, Jordan K. Terry, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Jin Shen Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[13] Shuhe Wang, Shengyu Zhang, Jie Zhang, Runyi Hu, Xiaoya Li, Tianwei Zhang, Jiwei Li, Fei Wu, Guoyin Wang, and Eduard H. Hovy. Reinforcement learning enhanced llms: A survey, 2024.