The Dark Side of VLM Rewards: Understanding and Mitigating False Positive Noise

Anonymous Author(s)
Affiliation
Address
email

Abstract: Vision-Language Models (VLMs) have shown promise as reward models for reinforcement learning (RL) agents, potentially providing rich reward signals based on how well an agent's trajectory aligns with expert instructions. However, our observations reveal that RL agents trained with VLM rewards often underperform compared to those employing only intrinsic (exploration-driven) rewards, contradicting expectations set by recent work. To understand this unexpected outcome, we analyze the noisy reward signals from VLMs, particularly focusing on the prevalence and consequences of false positives and false negatives. Our analysis revealed that false positive rewards of VLMs were a major factor contributing to the poor learning efficacy. To address this, we developed a novel reward function, BIMI, that prioritizes minimizing false positive rewards while accepting a slight increase in false negatives as a necessary trade-off. Experiments demonstrate that BIMI substantially improves agent performance across challenging environments, highlighting the critical importance of addressing reward noise when applying VLM-based rewards to complex real-world scenarios.

Keywords: RL, VLM-based Reward Model, Sparse Reward Environments

1 Introduction

Natural language instructions are increasingly recognized as a valuable source of reward signals for reinforcement learning (RL) agents to learn complex tasks. In particular, a growing trend in agent learning involves using vision-language models (VLMs) for reward modeling. This approach measures the semantic similarity – often quantified by cosine similarity – between the embedding representations of an agent's behaviors (i.e., past trajectories) and the provided instructions, all within the same embedding space [1, 2, 3, 4].

However, we observed that RL agents trained with VLM reward models, while effective in simplified settings, often struggled with tasks involving complex dynamics and longer action horizons. This is evident in several recent works – for instance, Goyal et al. [2] reported the effective use of VLM rewards in Montezuma's Revenge, a notoriously challenging Atari game. However, we observed that this success was confined to individual sub-tasks and the agent struggled when attempting to scale up to the full game. Similarly, Du et al. [4] demonstrated impressive performance of VLM rewards in guiding agents within a 2D survival game. However, their study was conducted in a modified environment with a reduced observation and action space using internal game state information and manually defined macro actions. Consequently, when tested in the original, unmodified environment, their agent's performance did not exceed that of agents using only *intrinsic* (exploration-driven) rewards.

The consistent underperformance of VLM-based models in complex environments, **particularly their unexpected failure to outperform even simple exploration-driven rewards**, raised significant concerns about their reliability in real-world applications. This discrepancy, where VLM

Submitted to the 8th Conference on Robot Learning (CoRL 2024). Do not distribute.

reward models underperformed contrary to their perceived potential, prompted us to investigate the underlying causes of this performance gap. Our findings indicate that **noisy reward estimates** in VLMs are a key factor contributing to poor learning efficacy. We attribute this noise primarily to the approximation errors inherent in the *cosine similarity metric* commonly used in these models. Specifically, our analysis focuses on two types of noise: false positives, which involve rewarding unintended behaviors, and false negatives, which occur when correct behaviors are not rewarded.

We posit that false positive reward noise (see Figure 1) are not only more prevalent but potentially more detrimental to the learning process than false negatives, a hypothesis supported by our empirical and theoretical findings. To this end, we propose a novel reward function, BIMI (**Bi**nary **M**utual **I**nformation Reward). It leverages binary reward signals to directly reduce the occurrence of false positives and incorporates a mutual information term to prevent overfitting to potentially noisy signal sources. Our experiments demonstrate that BIMI significantly improves the learning efficacy of agents trained by VLM rewards across various challenging environments.

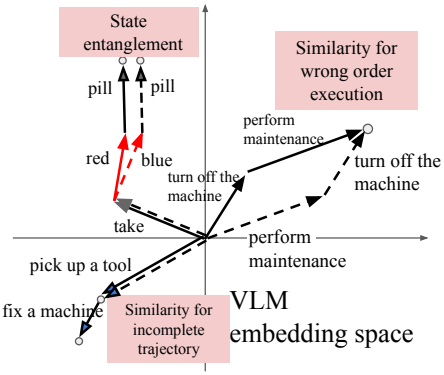


Figure 1: Schematic diagram of false positives in a VLM embedding space. The unintended agent's trajectory (dashed line) may exhibit high cosine similarity to the instruction (solid line) in the embedding space, as indicated by the proximity of their endpoints in the embedding space. Despite this apparent similarity, the unintended trajectories fundamentally fail to fulfill the intended instruction. Therefore, the rewards guides the agent towards incorrect behaviors. The figure shows three distinct cases of false positives. Refer to Section 4 for more details.

2 Related Work

Use of VLMs as Reward Models. VLMs have been pivotal in robotics studies, serving as reward models that guide agents to follow instructions [5, 6, 7]. While most research primarily focuses on leveraging VLMs to overcome the challenge of manual reward design for complex tasks [8], the impact of reward noise and its implications for policy convergence rates are often overlooked. As mentioned in Section 1, some work sidesteps the noisy reward problem by accessing internal state information from the game engine as well as providing predefined action macros [4, 9], thereby preventing the accumulation of noise over longer horizon. However, understanding the impact of reward noise from VLMs is crucial for developing reliable language interface for RL agents, as it directly affects their ability to learn effectively in real-world tasks.

Mitigating Reward Model Failures. Research by Ghosal et al. [10] and Fu et al. [11] has introduced methods to counteract unreliable reward signals from learned VLMs. These strategies involve employing a parallel exploration-based policy alongside the reward-maximizing policy, thereby reducing reliance on potentially misspecified VLM rewards. Our work contributes to this growing body of research by proposing a novel reward function that directly addresses the issue of noisy rewards in VLM-based models, complementing approaches that use exploration policies to escape local optima. Furthermore, we have tested the synergy of combining our reward function with exploration strategies, demonstrating how these approaches can be integrated for improved performance.

3 Formal Problem Statement

In the context of using instructions as auxiliary guide for RL agents, we frame our task as a MDP defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, r^e, \gamma \rangle$, where $\mathcal{S}$ represents a set of states $s \in \mathcal{S}$, $\mathcal{A}$ represents a set of actions $a \in \mathcal{A}$, and $\mathcal{T}(s' \mid s, a)$ describes the dynamics of the environment. $s_0 \in \mathcal{S}$ is the initial state, and $\gamma \in [0, 1]$ is the reward discount factor. $r^e(s, a, s')$ is the environmental reward function.

In this work, we focus on a sparse reward setting, where the agent receives a non-zero reward only when reaching goal states $S_G \subset \mathcal{S}$, and 0 otherwise, with $|S_G| \ll |\mathcal{S}|$. An agent's trajectory is a sequence of states and actions $\tau = \langle s_t, a_t, \ldots, s_{t+T} \rangle$, where t is the initial time step and T is the length of the trajectory. The objective is to learn a policy π that maximizes the expected cumulative reward of the trajectories $\tau^\pi = \{\tau_1, ..., \tau_n \mid \tau_i \text{ induced by } \pi\}$.

A sparse reward function only *realizes* a set of acceptable policies, rather than distinguishing between them more finely (as derived from Abel et al. [12]; see Appendix A.1 for details). Building on this, we define the following:

(1) Policy Universe, Π*:* the set of all possible policies for the MDP.

(2) Acceptable Policies, Π_G*:* the policies in Π_G are those policies whose start-state values $V^\pi(s_0)$ are within a small ϵ of the optimal value, while all other policies have strictly lower values.

(3) Unacceptable Policies, Π_B*:* formally, $\Pi_B = \Pi \setminus \Pi_G$, which refers to the set of policies that do not meet the near-optimal criteria and are therefore considered not acceptable.

A task instruction L is a sequence of natural language sentences $L = \{l_1, l_2, \ldots, l_n\}$, where each l_k induces a set of desired sub-trajectories τ^{l_k}. A complete task execution is represented by a trajectory that concatenates instructions' corresponding sub-trajectories in order, traversing from the initial state s_0 to a goal state in S_G.

An auxiliary VLM-based reward model provides more frequent rewards by evaluating the semantic similarity between the agent's trajectory and the specified instruction sentence. This can be formally represented as $r^v(\tau, l_k) = p(l_k|\tau)$ (see Figure 2), where r^v denotes the VLM-based reward function. The use of *non-Markovian* reward functions in MDP environments has been well-established in the literature, particularly through the work on reward machines [13, 14]. Such a VLM-based reward function consists of two components: an embedding encoder $\phi(\cdot)$ and a similarity function. The reward is calculated as the cosine similarity between the embeddings, which can be expressed as $\frac{\phi(\tau)\cdot\phi(l_k)}{\|\phi(\tau)\|\|\phi(l_k)\|}$.

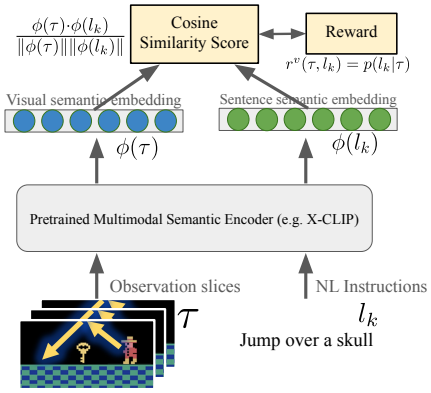


Figure 2: Illustration of reward signals from a vision-language reward model

Reduction of Convergence Rate. To understand the effects of reward noise on policy learning, it's crucial to analyze its impact on the convergence rate. In sparse reward settings, the convergence rate – defined as the number of training iterations needed to learn an ϵ-optimal policy – can be significantly affected by false positive rewards that are incorrectly given for policies that do not lead to the goal states.

To illustrate this, we analyze the convergence rate by measuring the probability of learning acceptable policies ($P(\pi \in \Pi_G)$) over each update iteration. The following theorem demonstrates how false positive rewards impact the convergence rate:

Theorem 3.1 (Convergence rate reduction)**.** *Actor-Critic algorithm, gradient ascent on* $Q(s,a)\pi_{\theta_i}(a|s)$ *pushes the next updated policy* $\pi_{\theta_{i+1}}$ *in the direction provided by the Q value function. In the direction of maximizing* $P(\pi \in \Pi_G)$*, the gradient of* $Q(s,a)\pi(a|s)$ *can be expressed as follows:*

$$\nabla_\theta Q_\phi(s,a)\pi_\theta(a|s) = const \cdot \mathbb{E}[G^{\pi\in\Pi_B}]\nabla_\theta\pi_\theta(a|s) + (\mathbb{E}[G^{\pi\in\Pi_G}] - \mathbb{E}[G^{\pi\in\Pi_B}])P(\pi \in \Pi_G)\nabla_\theta\pi_\theta(a|s)$$

where Π_G *is the set of acceptable policies;* Π_B *is the set of unacceptable policies that follow instructions but fail to reach the goal;* $\mathbb{E}[G^\pi]$ *is expectation of the cumulative rewards by executing policy* π *in one episode.*

The proof is in Appendix A.2. In this context, false positive rewards correspond to rewards assigned to policies within the set Π_B, which represents undesirable policies. Therefore, $\mathbb{E}[G^{\pi \in \Pi_B}]$ can be interpreted as the expected cumulative false positive rewards. This theorem highlights how the presence of $\mathbb{E}[G^{\pi \in \Pi_B}]$ reduce the convergence rate by: (1) introducing deviation in the policy update directions (the first term), and (2) reducing the gradient of the desired policy update (the second term). Both effects can significantly hinder the convergence process.

The issue of noisy rewards is not merely a theoretical concern; it has practical implications for the effectiveness of instruction-guided RL in real-world applications. In the next section, we will investigate how cosine similarity metric used in VLM-based reward models inevitably introduce false positive rewards. Additionally, we will empirically evaluate the impact of both false positives and false negatives on the learning efficacy of agents in sparse reward environments.

4 Approximation Error of Cosine Similarity

In this section, we will discuss two fundamental issues associated with cosine similarity scores in RL contexts: *state entanglement* and *composition insensitivity*. The former issue, state entanglement, refers to the metric's inability to recognize trajectories $\{\tau_1, ..., \tau_n \mid \tau_i \text{ induced by } \pi \in \Pi_B\}$ that, while being cosine similar to the target instruction in the embedding space, fail to reach the goal states in S_G. The latter issue refers to the metric's tendency to reward trajectories that execute the sub-tasks (corresponding to individual instruction sentences l_k) in an incorrect order. This is problematic in RL tasks where the specific sequence of executing sub-tasks, as defined by the order of instructions in $L = \{l_1, l_2, \ldots, l_n\}$, is crucial for reaching the goal states.

The Issue of State Entanglement State entanglement refers to the issue where the cosine similarity metric erroneously pays more attention to lexical-level similarity while lacking comprehension of the underlying state transitions. Consequently, rewards are given to trajectory-instruction pairs that are cosine similar in embedding space but in fact result in distinct state transitions. For instance, consider the significant contrast between "take the *red* pill" and "take the *blue* pill". Despite their lexical similarity, they lead to vastly different states. However, the cosine similarity metric may represent them as similar due to the shared action "take" and shared object "pill", disregarding the critical difference in state outcomes. Understanding state transitions is crucial in sequential decision-making scenarios. Otherwise, rewards may be given to trajectories that lead to unintended states, resulting in the learning of unacceptable policies.

The Issue of Composition Insensitivity Composition insensitivity in cosine similarity metrics gives rise to two issues: (1) *rewarding incomplete task execution* – cosine similarity may incorrectly reward partial task completion, as even incomplete trajectories can receive high similarity score in the embedding space. For instance, in a task to "pick up a tool, then fix a machine," the model might prematurely reward the agent for merely picking up the tool, neglecting the crucial repair action. This can lead to suboptimal learning, with the agent potentially getting stuck in easier, incomplete subtasks rather than fully accomplishing the task. (2) *insensitivity to the ordering of execution* – cosine similarity often fails to adequately penalize incorrect execution sequences. In a safety protocol requiring an agent to "turn off the machinery, then perform maintenance," the metric might assign high rewards based merely on the presence of relevant actions, disregarding their order. While some large language models have become sensitive to the order of tokens [15], the compact visual and sentence embeddings from multimodal VLMs remains largely insensitive to sequential information [16]. This limitation can lead to potentially dangerous outcomes in scenarios where execution sequence is critical, such as in safety protocols or complex manufacturing processes.

Figure 1 illustrates various scenarios where high similarity scores are erroneously assigned to false positive cases. To empirically demonstrate the issue, Section 4.1 presents experiments on false positive rewards and their impact on agent learning in sparse reward environments.

4.1 Experiments on Reward Noise Impact

Our experiments test the following hypothesis: **(H1)** The two issues of *state entanglement* and *composition insensitivity* exist; **(H2)** *false positive* rewards are prevalent during training; **(H3)** similarity-based reward models lacking noise handling mechanisms underperform against strong intrinsic reward models in sparse reward environments; **(H4)** *false negatives* may not be as harmful as *false positives*.

Setup. We evaluate these hypotheses through various challenging sparse-reward environments: (1) *Crafter*, an open-ended 2D Minecraft environment [17]; (2) *Montezuma's Revenge*, a classic hard adventure game in Atari [18]; and (3) *Minigrid 'Go To Seq'*, a hard task involving long-horizon navigation and object interactions [19]. Two cosine similarity-based reward models were tested: (1) *Pixl2R* by Goyal et al. [3], which uses only the current video frame to determine if the goal state specified in the instruction has been reached; and (2) *ELLM-*, a variant of *ELLM* by Du et al. [4]. Unlike ELLM, which queries instructions from LLMs in real-time, ELLM- directly uses preset expert instructions and compares them with the transition differences of the agent's trajectory. The VLM backbones used are: (1) *CLIP* [20], pretrained by image-text pairs; and (2) *X-CLIP* [21], pretrained by video-text pairs. To ensure high-quality finetuning data, we used internal information from the game engine to annotate expert trajectories from expert agents. To demonstrate how noisy reward signals hinder learning, we selected a strong intrinsic reward model *DEIR* [22] for comparison. It provides auxiliary rewards based on observation novelty to encourage exploration. See Appendix A.3 for detailed implementation of the experiments.

Evaluation Metric. We used the ***score*** metric adapted from the Crafter benchmark [17] for performance evaluation, as it better captures an agent's consistent performance across multiple subtasks. This metric follows the intuitive 'higher is better' principle. Unlike *maximum total rewards* metric, which fail to capture consistent performance in sparse reward settings, the score metric provides a more reliable measure of learning progress.

Reward Noise Issue. To investigate **H1**, we evaluated the models' sensitivity by examining how cosine similarity scores change for manipulated trajectory-instruction pairs. The state entanglement test involved reversing trajectories and negating instructions (i.e., "do not do l_k"). The composition insensitivity test examined concatenated pairs of trajectory-instruction data. For example, given (τ_1, l_1) and (τ_2, l_2), we create a concatenated pair $(\tau_1 + \tau_2, l_1 + l_2)$. We then test two types of manipulations – (1) swapping the order within one modality: e.g., $(\tau_2 + \tau_1, l_1 + l_2)$; and (2) truncating one modality: e.g., $(\tau_1, l_1 + l_2)$. Overall, we categorized the trajectory-instruction pairs for evaluation into three types: (1) matched pairs – these are positive examples where the trajectory correctly corresponds to the given instruction. (2) not-matched pairs – these are negative examples where the trajectory does not correspond to the given instruction. (3) manipulated pairs: derived from matched pairs by altering either the trajectory or the instruction. Ideally, manipulated pairs should receive low

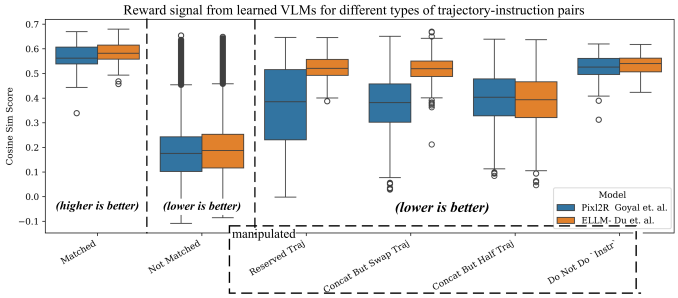


Figure 3: Learned VLM models differentiate between matched and not-matched pairs, but struggle with O.O.D. cases. They incorrectly assign high scores to manipulated pairs, which should be low as the trajectories in the manipulated pairs fail the instruction

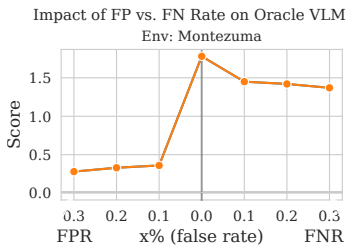


Figure 4: The false positive vs. false negative oracle model. The false positive model get a more severe drop in the final training score.

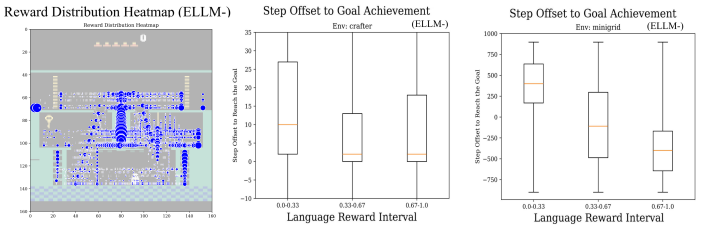


Figure 5: The heatmap shows the cumulative rewards received at various locations, with larger circle sizes indicating higher rewards. The later figures shows the offsets between the state where rewards are given and the actual goal-reaching state. Agents are getting both issues of false positives and false negatives during training

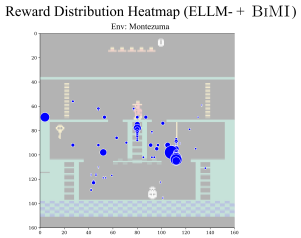


Figure 6: The ratio of false positive rewards is significantly reduced after applying BIMI

similarity scores as essentially the trajectory are not fulfilling the instruction. However, our results reveal that the reward model assigns high scores to these manipulated pairs, particularly in the "do not do l_k" case. This finding highlights the noise issue in cosine similarity-based reward models (see Figure 3). It's worth noting that the poor performance in the negation case aligns with broader challenges in natural language processing. Recent studies [23, 24] have highlighted that negation is central to language understanding but is not properly captured by modern language models. This limitation extends to VLMs and significantly impacts their ability to provide accurate rewards in complex scenarios and instructions.

Prevalence of False Positives. To address **H2**, we analyzed reward distribution heatmap from VLM-based reward models during training. The heatmap revealed a concerning trend: RL agents engage in reward hacking, receiving rewards across vast areas of the environment rather than just at goal states. For instance, in *Montezuma* environment where the goal is to grab the key and escape the room, we observed that agents received rewards even for falling off cliffs, which undoubtedly contribute to the detrimental $\mathbb{E}[G^{\pi \in \Pi_B}]$ specified in Theorem 3.1. For environments without fixed camera views, we calculated the step offset between the current rewarded state and the actual goal state. A positive offset indicates that a reward was given before reaching the goal state, signifying a false positive reward, while a negative offset suggests that the agent reached the goal state but the reward model failed to recognize it, indicating a false negative reward (see Figure 5). Interestingly, we observed a large amount of negative step offsets in Minigrid environments. We attribute this to Minigrid's abstract shape-based visual representations, which fall outside the VLM's pretraining distribution.

Figure 7: Score metric across environments (equivalent to total rewards, higher is better). $\star$ denotes baseline intrinsic reward model. VLM reward models without noise handling underperformed. All models are based on PPO.

Models	Type	Monte.	Minigrid	Crafter	% vs. DEIR
PPO	Pure	0.151	24.993	16.863	-28%
DEIR $\star$	Intrinsic	0.174	55.556	19.758	–
Pixl2R	VLM	0.142	12.422	9.409	$-\mathbf{49}\%$
ELLM-	VLM	0.150	19.406	10.826	-41%
Pixl2R + DEIR	VLM + intr.	0.176	17.372	10.440	-38%
ELLM- + DEIR	VLM + intr.	0.178	30.985	11.857	-27%

Impact on Learning. We trained agents using learned VLM reward models and compared their learning efficacy against intrinsic reward models. As shown in Table 7, our results confirmed **H3**: *instruction-guided RL agents using learned VLM reward models without noise handling consistently underperform compared to DEIR, the intrinsic reward-based RL agent.* To investigate the impact of false negatives versus false positives (**H4**), we designed an oracle Pixl2R model with two variants: a false negative model and a false positive model. The false negative model only rewards the agent for reaching subgoal states described in the instruction, with a probability of x% that some rewarding states in the map are removed. In contrast, the false positive model rewards the agent for reaching every subgoal, but also introduces a 0.1 one-off reward for certain locations, covering x% of the map. The results indicate that false negatives may be less detrimental to agent performance than false positives (see Figure 4). This performance difference can be explained through our theoretical framework:

(1) False negative models maintain the coefficient $\mathbb{E}[G^{\pi\in\Pi_G}] - \mathbb{E}[G^{\pi\in\Pi_B}]$, ensuring steady gradient ascent towards maximizing $P(\pi \in \Pi_G)$;

(2) By decreasing $\mathbb{E}[G^{\pi\in\Pi_B}]$, false negative models minimize deviations from the target direction, leading to more stable learning.

In contrast, scenarios with high $\mathbb{E}[G^{\pi\in\Pi_B}]$ (i.e., the false positive case) significantly reduce the gradient ascent rate and also introduce deviations from the target direction. The findings thus challenged the prevailing assumption that false negatives are the primary issue in the degeneracy of instruction-guided RL. See Appendix A.4 for more experimental details.

5 Solution to the Reward Noise Issue

5.1 Binary Signal and Conformal Prediction Thresholding

Our experiments have shown that the issue of false positives may be more detrimental to learning than false negatives. Based on these findings, we we propose a reward function that provides one-time binary rewards only when the similarity between the agent's current trajectory and the instruction exceeds a high confidence threshold. This method effectively reduces the likelihood of rewarding unacceptable trajectories. To achieve this, we introduce a thresholding mechanism using a calibration set of true positive trajectory-instruction pairs. This threshold, denoted as $\hat{q}$, is set to the empirical quantile of cosine similarity scores at the significance level $1 - \alpha$. Pairs whose similarity scores fall below this threshold $\hat{q}$ receive no reward. Conversely, pairs exceeding $\hat{q}$ receive a one-time $+1$ reward, i.e., $r^v_{\text{BI}}(\tau, l_k) = \mathbf{1}_{\{p(l_k|\tau)\geq\hat{q}\}}$. This approach statistically guarantees a high probability (at least $1 - \alpha$) that true positive pairs are recognized while minimizing frequency of false positives errors [25]. See Appendix A.5 for detailed threshold calculation.

5.2 Mutual Information Maximization

Intuitively, when we observe rewards coming from a particular signal source too frequently, we tend to downplay the significance of that signal to avoid over-reliance. This intuition is effectively captured by incorporating a *mutual information maximization* term into the reward function. Specifically, the updated reward function $r^v_{\text{MI}}(\tau, l_k)$ measures the mutual information between the agent's trajectory and the instruction. Mathematically, it can be expressed as:

$$\begin{aligned} r^v_{\text{MI}}(\tau, l_k) &= I(l_k; \tau) = D_{KL}(p(l_k, \tau) \,||\, p(l_k)p(\tau)) \\ &= \mathbb{E}_{\tau\sim\pi_\theta, l_k\sim L}[\log p(l_k \mid \tau) - \log p(l_k)] \end{aligned} \tag{1}$$

where $p(l_k \mid \tau)$ comes from the similarity score provided by VLMs, and $p(l_k)$ is the frequency of the instruction l_k being fulfilled by the agent's trajectory. Therefore, the second term in the equation serves as a regularization term that downplays the significance of the reward signal when it is too frequent. For instance, if a VLM frequently detects that the agent's actions are fulfilling the "climbing the ladder" instruction, even when the agent is performing unrelated tasks, the reward signal for this instruction will be downplayed. $p(l_k)$ is calculated as follows:

$$p(l_k) = \mathbb{E}_{\tau\sim\pi_{\theta_{-1}}}\left[\frac{\sum_{t=1}^{T_\tau} \mathbf{1}_{\{p(l_k|\tau_t)\geq\hat{q}\}}}{T_\tau}\right] \tag{2}$$

Here, τ_t is the agent's trajectory up to time t, and if the VLM deems the trajectory as fulfilling the instruction (i.e., $p(l_k \mid \tau_t) \geq \hat{q}$), we increment the count. Dividing the count by the total trajectory length T_τ gives the empirical frequency of the instruction being fulfilled. The subscript θ_{-1} in $\pi_{\theta_{-1}}$ indicates that the trajectories are sourced from rollouts in the previous policy iteration, acknowledging the impracticality of real-time computation of $p(l_k)$ during an ongoing episode.

To enhance the stability of the training process, we adopt a linearized version of the mutual information maximization approach, as proposed by Li et al. [26]. Overall, BIMI, the proposed reward

function that enhances the noise resilience of VLM-based reward models, can be expressed as follows:

$$r^{v}_{\text{BIMI}}(\tau, l_k) = \max(\mathbf{1}_{\{p(l_k|\tau)\geq\hat{q}\}} - p(l_k), 0) \tag{3}$$

It's important to note that the BIMI approach primarily mitigates false positives (FP) rather than false negatives (FN). By implementing a high confidence threshold and a binary reward signal, we effectively reduce the likelihood of rewarding incorrect trajectories, thus addressing the FP issue. While this conservative approach may potentially increase FNs, as some correct trajectories might fall below the threshold, our experiments indicate that this trade-off is beneficial. We found that false positives are more detrimental to learning in instruction-guided RL tasks, justifying our preference for a method that prioritizes FP reduction.

6 Experiments

We evaluate the BIMI reward function using baseline agents (*Pixl2R* and *ELLM-*) and their BIMI-enhanced counterparts, while also exploring potential synergies with the intrinsic reward model DEIR. We follow the same experimental setup as in Section 4.1, with additional details in Appendix A.6.

Table 1: Model score across various environments. $\star$ is the baseline agents with a learned VLM-based reward model to compare with. BIMI significantly improves performance in *Montezuma* and *Minigrid*, while showing mixed results in *Crafter* due to task-specific characteristics

Methods	Montezuma	% vs. $\star$	Minigrid	% vs. $\star$	Crafter	% vs. $\star$
Pixl2R $\star$	0.142 ± 0.003	–	12.422 ± 2.439	–	9.409 ± 0.022	–
Pixl2R + Bi	0.137 ± 0.009	-3.5%	31.236 ± 2.040	151%	10.735 ± 0.784	14%
Pixl2R + BiMI	0.162 ± 0.022	14%	37.507 ± 7.832	199%	7.951 ± 0.351	-15%
Pixl2R + DEIR	0.176 ± 0.009	23%	17.372 ± 0.514	39%	10.440 ± 1.015	10%
Pixl2R + BiMI + DEIR	0.267 ± 0.016	88%	57.759 ± 2.157	364%	11.014 ± 0.190	17%
ELLM- $\star$	0.150 ± 0.004	–	19.406 ± 10.067	–	10.826 ± 1.017	–
ELLM- + Bi	0.151 ± 0.016	0.6%	29.788 ± 1.290	53%	11.175 ± 0.601	3.2%
ELLM- + BiMI	0.156 ± 0.014	4.0%	33.683 ± 3.990	74%	9.425 ± 0.267	-12%
ELLM + DEIR	0.178 ± 0.029	20%	30.985 ± 3.507	59%	11.857 ± 1.152	9.5%
ELLM- + BiMI + DEIR	0.279 ± 0.078	86%	56.281 ± 6.193	190%	13.170 ± 0.393	22%

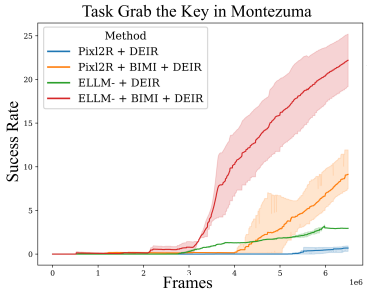


Figure 8: BIMI reward showed faster and higher success rates on difficult tasks in Montezuma

6.1 Overall Performance and Ablation Study

The overall improvements were substantial. As shown in Table 1, BIMI led to a 67% improvement for *Pixl2R* and a 22% improvement for *ELLM-*. These results are also illustrated in Figure 8 Figure 13 in Appendix A.7. Our ablation study highlights the distinct contributions of the binary reward (BI) and Mutual Information (MI) components within the BIMI framework. The binary reward mechanism alone accounted for a substantial 36.5% improvement in performance. When excluding the results from *Crafter*, MI component further contributes a 23% improvement over the binary reward alone. Please see Appendix A.7 for result details for each test environment.

7 Conclusion

Our research reveals two key findings in agent learning using VLM-based reward models: (1) false positive rewards, rather than false negatives, are more detrimental to policy learning; and (2) our proposed BIMI reward function significantly mitigates the slowdown in learning caused by false positives, thereby paving the way for more reliable and effective "language as an interface" RL systems in practical settings.

Limitation. Our study primarily focused on linear sequences of language instructions, excluding more complex cases. Future research should investigate conditional and ambiguous instructions, which likely introduce additional challenges for VLM-based reward models. We also did not explore finetuning the VLM during agent training, a useful strategy as discussed by Fu et al. [11].

References

[1] R. Kaplan et al. Beating atari with natural language guided reinforcement learning. *ArXiv*, abs/1704.05539, 2017. URL https://api.semanticscholar.org/CorpusID:6022828.

[2] P. Goyal et al. Using natural language for reward shaping in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2019. URL https://api.semanticscholar.org/CorpusID:70350059.

[3] P. Goyal et al. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In *Language in Reinforcement Learning Workshop at ICML 2020*, 2020.

[4] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, 2023. URL https://api.semanticscholar.org/CorpusID:256846700.

[5] X. E. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y. fang Wang, W. Y. Wang, and L. Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6622–6631, 2018. URL https://api.semanticscholar.org/CorpusID:53735892.

[6] M. Shridhar et al. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR, 2022.

[7] P. Mahmoudieh et al. Zero-shot reward specification via grounded natural language. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:250340669.

[8] J. Clark. Faulty reward functions in the wild. https://openai.com/index/faulty-reward-functions/, 2016. Accessed: 2024-08-06.

[9] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang. Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents. In *Neural Information Processing Systems*, 2023. URL https://api.semanticscholar.org/CorpusID:268042457.

[10] G. R. Ghosal, M. Zurek, D. S. Brown, and A. D. Dragan. The effect of modeling human rationality level on learning rewards from multiple feedback types. In *AAAI Conference on Artificial Intelligence*, 2022. URL https://api.semanticscholar.org/CorpusID:251740992.

[11] Y. Fu, H. Zhang, D. Wu, W. Xu, and B. Boulet. Furl: Visual-language models as fuzzy rewards for reinforcement learning. In *Forty-first International Conference on Machine Learning*, 2024.

[12] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 34: 7799–7812, 2021.

[13] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2018. URL https://api.semanticscholar.org/CorpusID:51868784.

[14] J. Corazza et al. Reinforcement learning with stochastic reward machines. In *AAAI Conference on Artificial Intelligence*, 2022. URL https://api.semanticscholar.org/CorpusID:250297195.

[15] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[16] T. Pham, T. Bui, L. Mai, and A. Nguyen. Out of order: How important is the sequential order of words in a sentence in natural language understanding tasks? In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1145–1160, 2021.

[17] D. Hafner. Benchmarking the spectrum of agent capabilities. In *Deep RL Workshop NeurIPS 2021*, 2021.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

[19] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2018.

[20] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:231591445.

[21] Y. Ma, G. Xu, X. Sun, M. Yan, J. Zhang, and R. Ji. X-clip: End-to-end multi-grained contrastive learning for video-text retrieval. *Proceedings of the 30th ACM International Conference on Multimedia*, 2022. URL https://api.semanticscholar.org/CorpusID:250607505.

[22] T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian. Noveld: A simple yet effective exploration criterion. In *Neural Information Processing Systems*, 2021. URL https://api.semanticscholar.org/CorpusID:245877021.

[23] M. M. Hossain et al. An analysis of negation in natural language understanding corpora. *arXiv preprint arXiv:2203.08929*, 2022.

[24] T. H. Truong, T. Baldwin, K. Verspoor, and T. Cohn. Language models are not naysayers: an analysis of language models on negation benchmarks. *arXiv preprint arXiv:2306.08189*, 2023.

[25] M. Sadinle et al. Least ambiguous set-valued classifiers with bounded error levels. *Journal of the American Statistical Association*, 114(525):223–234, 2019.

[26] M. Li, X. Zhao, J. H. Lee, C. Weber, and S. Wermter. Internally rewarded reinforcement learning. In *International Conference on Machine Learning*, pages 20556–20574. PMLR, 2023.

[27] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *J. Mach. Learn. Res.*, 22(98): 1–76, 2021.

[28] K. Hornik et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[29] S. Moon, J. Yeom, B. Park, and H. O. Song. Discovering hierarchical achievements in reinforcement learning via contrastive learning. In *Neural Information Processing Systems*, 2023.

A Technical Appendix

Continued from the main text of *Overcoming Reward Model Noise in Instruction-Guided Reinforcement Learning*, the technical appendix consist of the following:

- **§A.1 Property of Sparse Reward Function**, which discuss why sparse reward function results in the categorization of policies into a set of acceptable policies and a set of unacceptable policies but not others.
- **§A.2 Proof of the Reduction of Convergence Rate**, which is referred in Section 3.
- **§A.3 Implementation Details of the Experiments**, which specify the implementation details for both the first and second stage experiments in Section 4.1 and Section 6 respectively.
- **§A.4 Additional Details of the Experiments of False Positive Rewards**, which is referred in Section 4.1.
- **§A.5 Pseudo-code for Empirical Quantile Calculation for Binary Signal Threshold**, which is referred in Section 5.1.
- **§A.6 Additional Implementation Details of the Experiments of BIMI Reward Function**, which is referred in Section 6.
- **§A.7 Detailed Experiment Results of BIMI Reward Function**, which provides detailed experiment results of BIMI reward function, which is referred in Section 6.

A.1 Sparse Reward Function and Range-SOAP

Formally, Abel et al. [12] defined that:

Theorem A.1 (Abel et al. [12])**.** *A reward function realizes a Range Set of Acceptable Policies (Range-SOAP)* Π_G *when there exists an* $\epsilon \geq 0$ *such that every* $\pi_g \in \Pi_G$ *is* ϵ*-optimal in start-state value,* $V^*(s_0) - V^{\pi_g}(s_0) \leq \epsilon$*, while all other policies are worse.*

When the reward signal is sparse, the agent only receives a reward upon reaching some goal states, and the reward function does not provide any feedback during the intermediate steps. We argue that the sparse reward function realizes a Range-SOAP, leading to the categorization of policies into acceptable and unacceptable policies.

Proposition A.2. *Sparse reward function "realizes" Range-SOAP (i.e., Range Set of Acceptable Policies).*

Justification:

A sparse reward function, where the agent only receives a reward upon completing the task, cannot prefer policies that lead to shorter task completion times. This is because either the agent completes the goal very quickly or slowly, they will receive nearly the same amount of cumulative rewards, and the reward function will not show a strong preference.

Since the sparse reward function does not induce a strict partial ordering on the policies, we say this reward function cannot realize a Partial Ordering (PO) task. Specifically, a PO on policies is a generalization of a Set of Acceptable Policies (SOAP) task. In a PO, the agent specifies a partial ordering on the policy space, where some policies are identified as "great", some as "good", and some as "bad" to strictly avoid, while remaining indifferent to the rest.

Therefore, the sparse reward function can realize a Set of Acceptable Policies (SOAP), where there is a set of policies that are all considered "good" or near-optimal, while all other policies are worse.

Furthermore, the sparse reward function will lead to a Range-SOAP, rather than an Equal-SOAP. Specifically, Equal-SOAP is a SOAP where all the acceptable policies are equally optimal in start-state value. This is because the good policies in the SOAP may differ slightly in their start-state values, as some may reach multiple goal states in the environment and thereby receiving different

cumulative rewards. Therefore, the sparse reward function will realize a Range-SOAP, where there is a range of acceptable policies that are all near-optimal in start-state value.

A.2 Proof of the Reduction of Convergence Rate

Specifically, the update rule of Actor-Critic algorithm is:

- **Critic**:

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi (\delta)^2 \tag{4}$$

  where

  $\delta = \mathbb{E}_{\pi_\theta}[G^{\pi_\theta} - Q_\phi(s,a)]$ is the Monte Carlo (MC) estimation error, $Q_\phi(s,a)$ is the Q-value function which measures the expected discounted cumulative reward given the state s and action a, and π_θ is the policy.

  G^{π_θ} is the rollout cumulative rewards from the trajectory τ^{π_θ} generated from π_θ.

- **Actor**:

$$\theta \leftarrow \theta + \alpha_\theta \frac{Q_\phi(s,a)\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \tag{5}$$

We need to make the following assumptions to simplify the theoretical analysis:

1. We use Π_G to represent the set of acceptable policies and use Π_B as an arbitrary set of "not acceptable" policies that are learned to follow instruction but fail to reach subgoals, i.e., $\Pi_B \cap \Pi_G = \varnothing$.

2. Assume the policy class parameterized by θ should be expressive enough to capture optimal or near-optimal policies, and the policy is initialized randomly from uniform distribution, i.e., $\pi_{\theta_0} \sim \mathcal{U}(\Pi)$. Meanwhile, the Q-value is initialized as $Q_{\phi_0}(s,a) = 0, \forall_{s\in S}\forall_{a\in A}$.

3. We assume that $|\Pi_G|$ and $|\Pi_B|$ is a fixed number predefined by the task environment while $\mathbb{E}[G^\pi|\pi \in \Pi_B]$ is non-zero as false positive rewards are unavoidable in real-world VLMs.

Since the update rule for Q-value is a gradient descent on $\|\mathbb{E}[G^{\pi_\theta} - Q_\phi(s,a)]\|^2$ and also we have that $\{\Pi_B, \Pi_G, \Pi \setminus (\Pi_B \cup \Pi_G)\}$ is a countable partition of the policy universe Π, the updated Q-value will approach as follows:

$$\begin{aligned}
&Q_\phi(s,a) \to \mathbb{E}[G^{\pi_\theta}] \\
&= P(\pi_\theta \in \Pi_B) \cdot \mathbb{E}[G^{\pi_\theta}|\pi_\theta \in \Pi_B] \\
&\quad + P(\pi_\theta \in \Pi_G) \cdot \mathbb{E}[G^{\pi_\theta}|\pi_\theta \in \Pi_G] \\
&\quad + P(\pi_\theta \in \Pi \setminus (\Pi_B \cup \Pi_G)) \\
&\quad \cdot \mathbb{E}[G^{\pi_\theta}|\pi_\theta \in \Pi \setminus (\Pi_B \cup \Pi_G)] \\
&= P(\pi_\theta \in \Pi_B) \cdot \mathbb{E}[G^{\pi_\theta}|\pi_\theta \in \Pi_B] \\
&\quad + P(\pi_\theta \in \Pi_G) \cdot \mathbb{E}[G^{\pi_\theta}|\pi_\theta \in \Pi_G]
\end{aligned} \tag{6}$$

Given that the update rule for policy π is the gradient ascent on $Q_\phi(s,a)\pi_\theta(a|s)$, we have the following:

$$
\begin{aligned}
&\nabla_\theta Q_\phi(s,a)\pi_\theta(a|s) \\
&= (\nabla_\theta P(\pi_{\theta_{old}} \in \Pi_B) \cdot \mathbb{E}[G^{\pi_{\theta_{old}}} | \pi_{\theta_{old}} \in \Pi_B]\pi_\theta(a|s)) \\
&\quad + (\nabla_\theta P(\pi_{\theta_{old}} \in \Pi_G) \\
&\quad \cdot \mathbb{E}[G^{\pi_{\theta_{old}}} | \pi_{\theta_{old}} \in \Pi_G]\pi_\theta(a|s)) \\
&= P(\Pi_B) \cdot \mathbb{E}[G^{\Pi_B}]\nabla_\theta \pi_\theta(a|s) \\
&\quad + P(\pi_G) \cdot \mathbb{E}[G^{\pi_G}]\nabla_\theta \pi_\theta(a|s) \\
&= (1 - P(\pi_G) - P(\pi_\theta \in \Pi \setminus (\Pi_B \cup \Pi_G))) \\
&\quad \cdot \mathbb{E}[G^{\Pi_B}]\nabla_\theta \pi_\theta(a|s) + P(\pi_G) \cdot \mathbb{E}[G^{\pi_G}]\nabla_\theta \pi_\theta(a|s) \\
&= \textit{const} \cdot \mathbb{E}[G^{\Pi_B}]\nabla_\theta \pi_\theta(a|s) \\
&\quad + (\mathbb{E}[G^{\pi_G}] - \mathbb{E}[G^{\Pi_B}])P(\pi_G)\nabla_\theta \pi_\theta(a|s)
\end{aligned} \tag{7}
$$

Justification of the assumptions.

- Regarding the non-zero probability of recovering the optimal policy at initialization, it is standard in theoretical analyses to assume a uniform distribution of a random variable at initialization (see Agarwal et al. [27]). This assumption does not contradict the conclusion about the convergence rate deterioration in Theorem 3.1.
- In reference to the realizability condition implied by Assumption 2, the expressiveness of the policy class parameterized by θ is an underlying assumption for deep learning models, supported by the The Universal Approximation Theorem [28].

Observation. Since the goal of the learning agent is to maximize $P(\pi \in \Pi_G)$ (i.e., to converge to an acceptable policy), we can see that the second term provides the target direction with rate $(\mathbb{E}[G^{\pi \in \Pi_G}] - \mathbb{E}[G^{\pi \in \Pi_B}])$. Therefore, the ascent rate will decrease when the cumulative rewards from trajectories that cannot reach the goal state (i.e., the false positive rewards) gets higher. Moreover, the first term $\textit{const} \cdot \mathbb{E}[G^{\Pi_B}]\nabla_\theta \pi_\theta(a|s)$ can be regarded as the deviation of target direction. It shows that the level of deviation is also positively proportional to the magnitude of rewards coming from failed trajectories. This theorem follows the intuition that the presence of false positive rewards can slow down the convergence rate of the learning agent.

A.3 Implementation Details of the Experiments

A.3.1 Environment Details

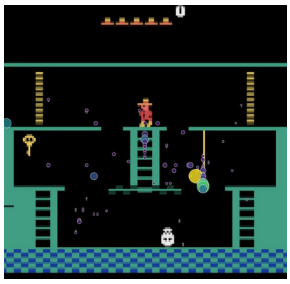
We describe each testing environment used in our experiments. More details introduction can be found in on the official project homepage of each benchmark [19, 18, 17].

- **Crafter** features randomly generated 2D worlds where the player needs to forage for food and water, find shelter to sleep, defend against monsters, collect materials, and build tools. The original Crafter environment does not have a clear goal trajectory or instructions; agents are aimed at surviving as long as possible and exploring the environment to unlock new crafting recipes. We modified the environment to include a preset linear sequence of instructions to guide the agent to mine diamond. However, this instruction was found to hinder the agent's performance. The nature of the task requires dynamic strategies and real-time decision-making, but the fixed instructions limited the agent. For example, the instruction did not account for what to do when the agent is attacked by zombies.
- **Montezuma's Revenge** is a classic adventure platform game where the player must navigate through a series of rooms to collect treasures and keys. The game is known for its sparse rewards and challenging exploration requirements. We manually annotate 97 instructions for the agent to follow, guiding it to conquer the game. The instructions were

designed to guide the agent through the game's key challenges, such as avoiding enemies, collecting keys, and unlocking doors.

- **Minigrid 'Go to seq' Task**: We use the 'Go to seq' task in the Minigrid environment, where the agent must navigate through a sequence of rooms and touch target objects in the correct order. This is a sparse reward task where the agent receives a reward of 1 only upon completing the entire sequence correctly. During the training phase, we randomly generate 50 different tasks, each with a room size of 5, 3 rows, and 3 columns. Each task features a unique room layout and target object sequence. The instruction complexity is set to 3, meaning there are at least 3 target objects to interact with in a specific order.

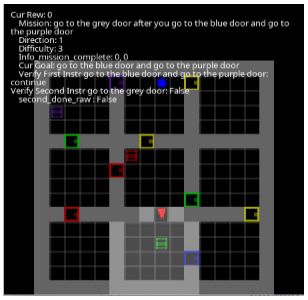
Montezuma and Instructions



Goal	Room	x	y
climb down the middle ladder	1	74	192
jump right to the yellow rope	1	109	199
jump right to the right platform	1	133	192
climb down the right ladder	1	133	148
jump over the skull to its left	1	76	151
climb up the left ladder	1	21	192
jump up to grab the key	1	13	209
jump left to the left roof	1	44	235

Figure 9: Illustration of the Montezuma's Revenge task. The agent must navigate through a series of rooms to collect treasures and keys.

Minigrid and Instructions



Goal
go to the grey door after you go to the blue door and go to the purple door
go to the box, then go to the yellow ball and go to a red door
go to the purple door and go to the purple door, then go to the red box
go to the green box, then go to a box
go to the ball and go to the yellow door after you go to a grey door
go to a ball, then go to the blue ball

Figure 10: Illustration of the Minigrid 'Go to seq' task. The agent must navigate through a sequence of rooms and touch target objects in the correct order.

Crafter and Instructions

Goal (Collect diamond)	
1. Collect wood	2. Collect wood again
3. Place table	4. Make wood pickaxe
5. Collect stone	6. Make stone pickaxe
7. Collect stone	8. Place furnace
9. Collect coal	10. Collect iron
11. Make iron pickaxe	12. Collect diamond

Figure 11: Illustration of the Crafter task. The agent must survive as long as possible and explore for new crafting recipes.

A.3.2 Instruction-Guide Procedure Details

The VLM-based reward model will have a pointer to the sequence of the instruction sentence, starting at the first sentence. For original models *Pixl2R* and *ELLM-*, we follow the setting in there original work where for each instruction sentence, the reward model will have a maximum cap of rewards (2.0) it can assign to the agent in one episode. When the cap is reached, the reward model will move its pointer to the next instruction sentence. For the BIMI reward model, the reward model will move its pointer to the next instruction sentence when the binary signal is triggered.

A.3.3 Finetuning VLM-based Reward Models

In contrast to previous work on Instruction-guided RL where they rely on hand-crafted oracle multi-modal reward models, we use actual pretrained VLMs to generate reward signals. 2 VLM backbone models are used in our experiments: 1) *CLIP* [20], pretrained by image-text pairs; and (2) *X-CLIP* [21], pretrained by video-text pairs. In particular, *Pixl2R* uses *CLIP* because it only uses the single latest frame as input. In contrast, *ELLM-* takes a slice of trajectory (i.e., multiple frames) as input, and thus uses either *X-CLIP* or *CLIP* with additional RNN encoder as the reward model.

Due to the cartoonish and abstract visuals of the testing environments, we need to further fine-tune the VLMs to adapt to this new visual domain. We use well-trained expert agents based on Moon et al. [29] to generate expert trajectories for the Crafter environments and annotate them with instructions using internal information from the game engine. For Minigrid environments, we use classical search-based planning robots to generate expert trajectories and annotate them with the corresponding task instructions. For Montezuma's Revenge, we manually annotate the expert trajectories.

For Minigrid and Crafter, we have 80,000 training pairs, while for Montezuma's Revenge, we have around 300 training pairs. These training data are of high quality, as we have made every effort to avoid false positive rewards due to poor training data quality. **To enhance our models' robustness, we also employed contrastive learning techniques during VLM training, utilizing similar manipulated data as hard negatives.** However, despite the fine-tuning process, false positive rewards remain unavoidable.

```
data_id,instruction,trajectory_chunk_file,trajectory_local_idx
0,climb down the middle ladder,montezuma/expert_traj_chunk_0.pkl,0
1,walk to the right side of the conveyor belt,montezuma/expert_traj_chunk_0
2,jump right to the yellow rope,montezuma/expert_traj_chunk_0.pkl,2
3,jump right to the right platform,montezuma/expert_traj_chunk_0.pkl,3
4,climb down the right ladder,montezuma/expert_traj_chunk_0.pkl,4
5,jump over the skull,montezuma/expert_traj_chunk_0.pkl,5
6,climb up the left ladder,montezuma/expert_traj_chunk_0.pkl,6
7,jump to grab the key,montezuma/expert_traj_chunk_0.pkl,7
8,jump left to the left roof ,montezuma/expert_traj_chunk_0.pkl,8
9,use key to open the left door,montezuma/expert_traj_chunk_0.pkl,9
10,walk left when the laser gate disappears,montezuma/expert_traj_chunk_0.p
11,walk to the middle when the laser gate disappears,montezuma/expert_traj_
12,wait until the laser gate disappears,montezuma/expert_traj_chunk_0.pkl,1
13,approach to the gem,montezuma/expert_traj_chunk_0.pkl,13
14,jump to grab the gem,montezuma/expert_traj_chunk_0.pkl,14
15,walk to the middle when the laser gate disappears,montezuma/expert_traj_
16,climb down the middle ladder,montezuma/expert_traj_chunk_0.pkl,16
17,wait until the spider goes away,montezuma/expert_traj_chunk_0.pkl,17
```

Figure 12: Example of training data for the Montezuma environment.

We used the threshold $\hat{q}$ introduced in Section 5.1 to make binary classification on the testing pairs to evaluate the performance of the fine-tuned VLM-based reward models. We found that VLM models had difficulty achieving high accuracy on Minigrid environment, which is likely due to the

too abstract and cartoonish nature of the environment, causing the VLMs to struggle to learn the visual-textual correspondence. We also found that X-CLIP did not perform better than CLIP in our experiments. We hypothesize that the cartoonish nature of the testing environments may have caused the X-CLIP model to struggle to learn the visual-textual correspondence. Thus, we used CLIP as the backbone model throughout our following experiments. The performance of the fine-tuned VLM-based reward models is shown in Table 2. **Even when the precision score reaches 0.98, indicating that only 2% of the rewards are false positives in the validation set, the agent can still significantly underperform in the testing environments. The core issue is that in out-of-distribution (O.O.D.) testing environments, false positive rewards are prevalent and inevitable. Therefore, it is crucial to design a reward function that is robust to reward noise.**

Table 2: Performance of fine-tuned VLM reward model on the testing dataset using the 90th percentile empirical quantile as threshold

Environment	Precision	Accuracy	F1 Score	Recall	Model
Crafter	0.9847	0.9466	0.8538	0.9702	CLIP ELLM-
Crafter	0.9799	0.9028	0.7618	0.9842	CLIP Pixl2R
Crafter	0.2095	0.2514	0.2868	0.9657	XCLIP ELLM-
Minigrid	0.7260	0.9200	0.7849	0.9763	CLIP ELLM-
Minigrid	0.6992	0.9086	0.7592	0.9616	CLIP Pixl2R
Minigrid	0.1716	0.2310	0.2642	0.9704	XCLIP ELLM-
Montezuma	0.8838	0.9638	0.8825	0.9478	CLIP ELLM-
Montezuma	0.8343	0.9108	0.7652	0.9842	CLIP Pixl2R
Montezuma	0.8044	0.9259	0.8045	0.9657	XCLIP ELLM-

A.3.4 Hyperparameters for Instruction-Guided RL Agents

In the experiments, all methods are implemented based on PPO with same model architecture. The Minigrid and Crafter environments use the same training hyperparameters as the Achievement Distillation paper [29]. For Montezuma's Revenge, we found that the performance of the agent was sensitive to the gamma and GAE lambda parameters. To improve the performance of agents in Montezuma's Revenge, we took two additional steps: (1) normalizing the observation inputs when computing the rewards, and (2) not normalizing the advantage during the GAE calculation. The hyperparameters are shown in the following tables.

Table 3: Model Parameters

Parameter	Value
model_cls	"PPORNNModel"
hidsize	1024
gru_layers	1
impala_kwargs	
- chans	[64, 128, 128]
- outsize	256
- nblock	2
- post_pool_groups	1
- init_norm_kwargs	
- batch_norm	false
- group_norm_groups	1
dense_init_norm_kwargs	
- layer_norm	true

Table 4: Crafter and Minigrid RL Parameters

Parameter	Value
gamma	0.95
gae_lambda	0.65
algorithm_cls	"PPOAlgorithm"
algorithm_kwargs	
- ppo_nepoch	3
- ppo_nbatch	8
- clip_param	0.2
- vf_loss_coef	0.5
- ent_coef	0.01
- lr	3.0e-4
- max_grad_norm	0.5
- aux_freq	8
- aux_nepoch	6
- pi_dist_coef	1.0
- vf_dist_coef	1.0

Table 5: Montezuma RL Training Parameters

Parameter	Value
gamma	0.99
gae_lambda	0.95
int_rew_type	"rnd"
pre_obs_norm_steps	50
algorithm_cls	"PPOAlgorithm"
algorithm_kwargs	
- update_proportion	0.25
- ppo_nepoch	3
- ppo_batch_size	256
- clip_param	0.1
- vf_loss_coef	0.5
- ent_coef	0.001
- lr	1.0e-4

A.4 Additional Details of the Experiments on the Impact of Noisy Rewards

Evaluation Metric Details In our experiments, we used a score metric adapted from the Crafter paper to evaluate agent performance across different environments. This score metric aggregates success rates for individual subtasks using a geometric mean. This metric was chosen over the *maximum total rewards* metric for several reasons:

1. **Consistency in Sparse Reward Settings:** Sparse reward environments often pose significant challenges for reinforcement learning agents. An agent might occasionally achieve high rewards by chance in one rollout but fail to replicate this success consistently in subsequent rollouts. This variability can lead to misleading evaluations if only the maximum total rewards are considered. The Score metric, by measuring the success rate of achieving each subgoal, provides a more stable and consistent measure of an agent's performance.
2. **Capturing Learning Stability:** The Score metric evaluates the agent's ability to consistently reproduce successful behaviors across multiple episodes. This is crucial in sparse reward settings, where the agent's performance can fluctuate significantly. By focusing on the success rates of individual subtasks, the Score metric offers a more granular and reliable assessment of the agent's learning progress and stability.
3. **Crafter Benchmark Standard:** The Crafter benchmark, which introduces the Score metric, is a well-regarded standard.

Crafter codebase provides *score* metric calculation by default. For Minigrid and Montezuma environments, we use the internal information from the game engine to detect whether the subtasks are completed, thus facilitating the calculation of the *score* metric.

A.4.1 Details of Manipulated Trajectory-Instruction Pairs to Evaluate Robustness

We evaluated the models' sensitivity by examining how cosine similarity scores change for manipulated trajectory-instruction pairs. These manipulations were designed to test the robustness of the models against various types of noise. Here's a detailed breakdown of the manipulations:

1. **Trajectory Reversal:** We inverted the sequence of frames within each trajectory (i.e., `frames = frame[::-1]`) to assess the model's ability to detect reversed state transitions. This manipulation tests whether the model can distinguish between forward and backward progression in the state transition.
2. **Instruction Negation:** We modified the original instructions by adding negation (e.g., changing "do l_k" to "do not do l_k" or "avoid l_k"). This tests the model's sensitivity to semantic changes in the instruction that fundamentally alter the goal.
3. **Instruction Rephrasing:** We rephrase the original instructions while maintaining their core meaning. This evaluates the model's robustness to linguistic variations and its ability to capture the essential semantic content of instructions.
4. **Concatenation and Order Swapping:** Given two trajectory-instruction pairs (τ_1, l_1) and (τ_2, l_2), we created concatenated pairs and then swapped the order in one modality. For example:
   - Original concatenation: $(\tau_1 + \tau_2, l_1 + l_2)$
   - Swapped trajectory: $(\tau_2 + \tau_1, l_1 + l_2)$
   - Swapped instruction: $(\tau_1 + \tau_2, l_2 + l_1)$

   This tests the model's sensitivity to the order of components in multi-step tasks.
5. **Concatenation with Partial Content:** We concatenated pairs but truncated one modality. For instance:
   - Truncated trajectory: $(\tau_1, l_1 + l_2)$
   - Truncated instruction: $(\tau_1 + \tau_2, l_1)$

   This assesses the model's ability to detect partial mismatches in longer sequences.

A.5 Pseudo-code for Empirical Quantile Calculation for Binary Signal Threshold

Using empirical quantile as threshold guarantees a high probability (at least $1 - \alpha$) that the true positive pairs are recognized while minimizing the average number of mistakes predicting false positives [25]:

Algorithm 1 Calculate Empirical Quantile ($\hat{q}$)

Require: Calibration set $\{\tau, l\}_n$, where l is the instruction sentence, τ is the corresponding trajectory, and n is the number of samples;
Significance level α;
VLM model reward model v

▷ *Obtain the similarity-based score* ◁
$\{r\}_n \leftarrow \{v(\tau, l)\}_n$
▷ *Compute the quantile level* ◁
$q_{\text{level}} \leftarrow \frac{\lceil (n-1) \times (1-\alpha) \rceil}{n}$
▷ *Compute the empirical quantile* ◁
$\hat{q} \leftarrow$ *np.quantile(*$\{r\}_n$*,* q_{level}*, method='lower')*
return $\hat{q}$

A.6 Implementation Details of the Experiments of BIMI Reward Function

We set confidence level for empirical quantile calculation to be $1 - \alpha = 0.9$. We adhered to the standard requirement of limiting the training budget to 1 million frames [17]. This constraint poses a significant challenge, particularly in sparse reward settings, as it demands that agents both explore efficiently and exploit their knowledge effectively within this limited budget.

To achieve the 1 million frame budget, we used the following configuration:

- `nproc`: 8 (Number of processes used for parallel environments)
- `nstep`: 512 (Length of the rollout stored in the buffer)
- `nepoch`: 250 (Number of epochs to train the RL policy)

This configuration results in approximately 1 million steps: 250 epochs $\times$ 512 steps $\times$ 8 processes $=$ 1,024,000 frames.

In the case of Montezuma's Revenge, we found that the 1 million frame limit used in Crafter was insufficient due to the game's complexity and sparse reward structure. To address this, we extended the training budget to 8 million frames. It's important to note that even with this increased frame count, agents were still unable to fully solve the task. As Zhang et al. [22] pointed out, about 1 billion frames are required to truly master Montezuma's Revenge. This vast difference in required training time (8 million vs 1 billion frames) underscores the exceptional difficulty of Montezuma's Revenge as a sparse reward task.

The implementation details for the BIMI reward function are consistent with those outlined in the first stage of experiments.

A.7 Detailed Experiment Results of BIMI Reward Function

A.7.1 Montezuma's Revenge

In *Montezuma*, Pixl2R+BIMI demonstrated an 14% performance increase compared to the original models (see Table 1), which is slightly below our expectations. We attribute this result to BIMI's intentional strategy of providing less frequent discrete rewards. While this strategy effectively reduces false positives, it does not substantially mitigate the inherent reward sparsity issue in *Montezuma*. However, **we discovered a remarkable synergy between BIMI and intrinsic reward models.** While previous models showed no significant improvements with *DEIR* (the intrinsic reward model) alone, combining BIMI and *DEIR* led to a 65% performance gain. The gap in collaboration effectiveness can be attributed to two factors. In the previous setup, the consistent presence of false positive rewards misled agents towards unacceptable behaviors and hindered further exploration. Now, BiMI's less frequent but more meaningful rewards provide anchor points for the agent's learning. Meanwhile, *DEIR*'s intrinsic rewards fill the gaps between these anchor points, encouraging the agent to explore efficiently in the interim.

As illustrated in Figure 6, BIMI rewards are now concentrated on key locations. A significant improvement is the minimal rewards given for falling off cliffs, which was a common source of false positives in the original model. Figure 8 demonstrates higher success rate grabbing the key in the first room, one of the most difficult task in *Montezuma*, highlighting the effectiveness of the proposed reward function and its collaboration with intrinsic reward models in guiding agents to solve difficult sparse-reward tasks.

A.7.2 Minigrid

ELLM-+BIMI achieved a remarkable 74% improvement in performance compared to the original models.This substantial gain is particularly noteworthy given the unique challenges presented by *Minigrid*. The abstract, shape-based visuals in *Minigrid* diverge drastically from the natural images used in VLMs' pretraining, preventing the models from effectively utilizing their prior pretraining

knowledge. Consequently, VLMs struggled to accurately assess similarities between Minigrid's abstract visuals and textual instructions, resulting in highly noisy reward signals. The significant improvement demonstrated by BIMI underscores its effectiveness in handling noisy signals, directly addressing our primary research challenge. This capability is crucial for deploying instruction-guided agents in real-world, unfamiliar scenarios, where visual inputs often deviate from the VLMs' training distribution, leading to noisy reward signals.

A.7.3 Crafter

We observed an intriguing pattern of results. The BI component alone led to 14% and 3.2% improvement in performance over the original models. However, contrary to our observations in other environments, the addition of the MI component actually decreased this improvement. This unexpected outcome can be attributed to the unique nature of *Crafter* task, where agents must repeatedly achieve the same subtasks (e.g., drinking water) for survival. The MI component, designed to discourage over-reliance on frequently occurring signals, inadvertently penalized the necessary repetition of survival-critical actions. Furthermore, note that instruction-guided RL agents, regardless of the reward model employed, were unable to outperform pure RL agents in *Crafter*. This discrepancy is due to the open-world nature of *Crafter*, which requires dynamic strategies and real-time decision-making that our testing setups did not fully capture. Despite these challenges, it is noteworthy that BI alone still managed to improve performance over vanilla VLM-based reward models, suggesting that reducing false positives is still beneficial across all testing environments. The combination of BIMI with *DEIR* (the intrinsic reward model) also showed promising results, indicating a productive balance between exploration (driven by *DEIR*) and exploitation (guided by BIMI instruction reward).

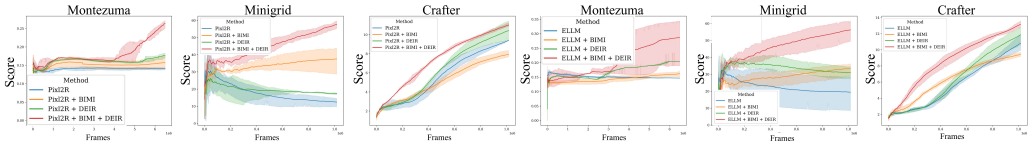


Figure 13: Besides the improvements of the score performance of agents across different environments with the BIMI reward function, it also collaborates well with intrinsic rewards. Combining both can lead to significant performance improvements

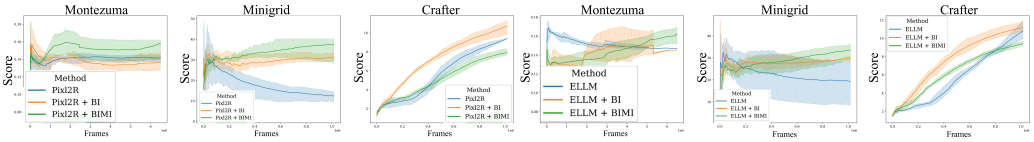


Figure 14: Ablation on the components of BIMI reward function. The binary reward (Bi) alone led to a 36.5% improvement compared to original models. Excluding Crafter, Mutual Information (MI) provided a 23% further improvement over Bi alone