The Dark Side of Rich Rewards: Understanding and Mitigating Noise in VLM Rewards

Sukai Huang[1], Shu-Wei Liu[2], Nir Lipovetzky[1] and Trevor Cohn[1*]
[1]School of Computing and Information Systems, The University of Melbourne, Australia
[2] Max Planck Institute for the Physics of Complex Systems, Nöthnitzer Strasse 38, 01187 Dresden, Germany
sukaih@student.unimelb.edu.au, sliu@pks.mpg.de, {nir.lipovetzky, trevor.cohn}@unimelb.edu.au

Abstract

While Vision-Language Models (VLMs) are increasingly used to generate reward signals for training embodied agents to follow instructions, our research reveals that agents guided by VLM rewards often underperform compared to those employing only intrinsic (exploration-driven) rewards, contradicting expectations set by recent work. We hypothesize that false positive rewards – instances where unintended trajectories are incorrectly rewarded – are more detrimental than false negatives. We confirmed this hypothesis, revealing that the widely used cosine similarity metric is prone to false positive estimates. To address this, we introduce BIMI (**Bi**nary **M**utual **I**nformation), a novel reward function designed to mitigate noise. BIMI significantly enhances learning efficiency across diverse and challenging embodied navigation environments. Our findings offer a nuanced understanding of how different types of reward noise impact agent learning and highlight the importance of addressing multimodal reward signal noise when training embodied agents[1].

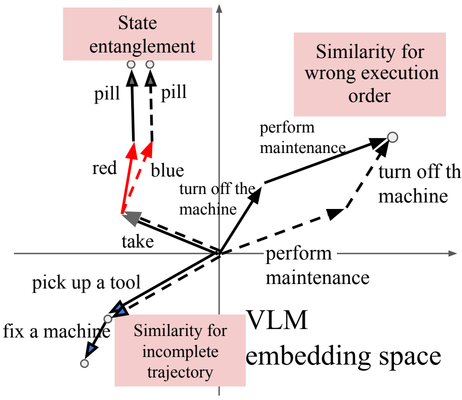


Figure 1: Schematic diagram of false positives in a VLM embedding space. The unintended agent's trajectory (dashed line) may exhibit high cosine similarity to the instruction (solid line) in the embedding space, as indicated by the proximity of their endpoints in the embedding space. Despite this apparent similarity, the unintended trajectory fails to fulfill the instruction. This illustrates how approximation errors can lead to false positive rewards. Refer to Section 5 for more details.

1 Introduction

Natural language instructions are increasingly recognized as a valuable source of reward signals for guiding embodied agents to learn complex tasks. In particular, a growing trend in embodied 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 (Kaplan et al., 2017; Goyal et al., 2019, 2020; Du et al., 2023).

However, we observed that embodied reinforcement learning (RL) agents trained with VLM rewards, 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. (2019) 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. (2023) 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 re-

[*] Now at Google DeepMind
[1] Code available at `https://shorturl.at/VsH70`

wards, **particularly their unexpected failure to outperform intrinsic rewards**, raised concerns about their reliability. This discrepancy, where VLM rewards 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. Specifically, our analysis centered around two classes of noise: false positives, which involve rewarding unintended trajectories, and false negatives, which occur when correct trajectories are not rewarded. We posit that false positive rewards (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. Among various sources of reward noise, our study particularly investigates the **approximation errors** within the commonly used cosine similarity metric. We examine how these errors generate false positive rewards, which in turn hinder learning.

To this end, we propose a novel reward function, BiMI (**Bi**nary **M**utual **I**nformation Reward). It uses binary reward signals to directly reduce the occurrence of false positives and incorporates a mutual information term to prevent overfitting to noisy signal sources. Our experiments demonstrate that BiMI significantly improves the learning efficacy of instruction following agents trained by VLM rewards across various challenging environments.

2 Related Work

Using VLMs as Reward Models. VLMs have been pivotal in robotics studies, serving as reward models that guide agents to follow instructions (Wang et al., 2018; Shridhar et al., 2022; Mahmoudieh et al., 2022). While most research primarily focuses on leveraging VLMs to overcome the challenge of manual reward design for complex tasks (Clark, 2016), 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 (Du et al., 2023; Wang et al., 2023), thereby preventing the accumulation of noise over longer horizon. However, understanding the impact of reward noise from VLMs is crucial for developing a reliable language interface for embodied learning agents in real-world applications.

Mitigating Reward Model Failures. Research by Ghosal et al. (2022) and Fu et al. (2024) has introduced methods to counteract unreliable rewards 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 mitigates the impact of false positive rewards from VLM-based models, complementing approaches that use exploration policies to escape local optima. Furthermore, we tested the synergy of combining our reward function with exploration strategies, demonstrating how these approaches can be integrated for further advancements.

3 Formal Problem Statement

We frame our task as an MDP defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_0, r^e, \gamma \rangle$, where $\mathcal{S}$ represents a set of states $s \in \mathcal{S}$, $\mathcal{A}$ represents a set of actions $a \in \mathcal{A}$, and $\mathcal{P}(s'|s,a)$ 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. $r^e(s,a)$ is the environmental reward function. An agent's trajectory is a sequence of states and actions $\tau_t = \langle s_0, a_0, \ldots, s_t \rangle$.

In this work, we focus on a sparse reward setting, where the agent receives a $+1$ reward only when reaching goal states $S_G \subset \mathcal{S}$, and 0 otherwise, with $|S_G| \ll |\mathcal{S}|$. This sparse reward setting motivates the use of expert instructions and VLMs to provide auxiliary reward signals for more effective RL. Specifically, we have a walkthrough L that breaks down a complex task into n expert-defined sub-tasks, each represented by a natural language instruction that is not necessarily atomic and can encompass multiple finer subgoals ($L = \{l_1, l_2, \ldots, l_n\}$). By following these sequential instructions, the agent can navigate from the initial state towards the goal states. A dedicated *non-Markovian* VLM-based reward model $r^v(\tau_t, l_{m(t)})$ is used to assess how well the agent's trajectory at current time t fulfills an instruction sentence $l_{m(t)}$. Here, $m(t)$ is a pointer at time step t that indicates the current instruction the agent is trying to complete[2]. The VLM provides auxiliary

[2] The use of *non-Markovian* reward functions in MDP has been well-established, particularly through the work on reward machines (Icarte et al., 2018; Corazza et al., 2022). For a complete evaluation, we also tested *Markovian* version of

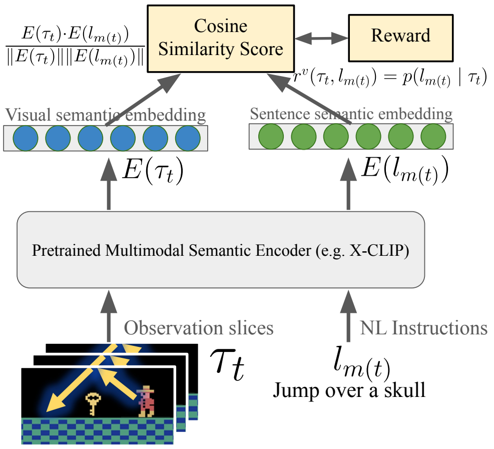

$\frac{E(\tau_t)\cdot E(l_{m(t)})}{\|E(\tau_t)\|\|E(l_{m(t)})\|}$
Cosine Similarity Score
Reward
$r^v(\tau_t, l_{m(t)}) = p(l_{m(t)} \mid \tau_t)$
Visual semantic embedding
Sentence semantic embedding
$E(\tau_t)$
$E(l_{m(t)})$
Pretrained Multimodal Semantic Encoder (e.g. X-CLIP)
Observation slices
NL Instructions
τ_t
$l_{m(t)}$
Jump over a skull

Figure 2: Illustration of reward from a VLM

rewards by evaluating the semantic similarity between τ_t and $l_{m(t)}$, as illustrated in Figure 2 (see Appendix A.1 for details).

4 Theoretical Analysis

In this section, we theoretically show how auxiliary rewards influence policy convergence. Initially, we establish that providing auxiliary rewards that reflect progress towards the goal state typically accelerates the convergence process compared to scenarios where only sparse environmental reward r^e is used. Then, we show that the presence of false positive rewards has a more detrimental effect on convergence than false negatives.

4.1 Auxiliary Rewards and Convergence

With sparse rewards, the gradient landscape is nearly flat, making gradient-based updates indistinguishable from a random walk in parameter space (Antognini and Sohl-Dickstein, 2018). In this problem, we are interested in $D := \|\theta_{goal} - \theta_0\|$ which is the distance in parameter space from initial parameters θ_0 to goal parameters θ_{goal}. We make the following assumption

Assumption 4.1. Expert knowledge can guide the parameter search along a path in parameter space, defined by a sequence of n intermediate parameter vectors $\theta_1, \ldots, \theta_n$, where each θ_i represents the parameters after learning sub-task l_i. As a result, the overall distance D can be decomposed into segments: $D \approx \sum_{k=1}^{n-1} d_i$, where $d_i = \|\theta_{i+1} - \theta_i\|$.

We also prove the following proposition

Proposition 4.2. *The sum of expected time for a series of random walks, each covering the shorter distance of an individual sub-task, is less than the expected time to travel the entire distance D in one long random walk:*

$$\frac{1}{n-1}\mathbb{E}[T_D] \leq \mathbb{E}\left[\sum_{i=1}^{n-1} T_{d_i}\right] < \mathbb{E}[T_D].$$

and thus show that subgoal-based auxiliary rewards improve the convergence of random walk optimization in a sparse reward landscape up to a factor of $(n-1)$ with details in Appendix A.2.1.

4.2 Connection to Heuristic-Guided RL

The problem and the possible solutions can be framed within the context of *Heuristic-Guided Reinforcement Learning* (HuRL) by Cheng et al. (2021). HuRL mandates that auxiliary reward signals serve as heuristics, where $h : \mathcal{S} \to \mathbb{R}$ approximates the future total rewards an agent expects to get starting from state s under the optimal policy π^* (i.e., $h(s) \approx V^*(s)$). These heuristics typically come from domain knowledge, aligning with expert instructions. We make the following assumption

Assumption 4.3. The fulfillment of $l_{m(t)}$ can be captured by intermediate goal states $\mathcal{S}^{m(t)}$.

This assumption is justified because often instructions in VLM come with end goals that can be indicated by definite states. For further discussions, please refer to Section 9. This assumption is required so that we can construct a domain-knowledge Markovian heuristic reward based on non-Markovian VLM rewards (see Appendix A.1.3 for the exact formula). With this construction, we are estimating the possible reward in the future by evaluating how well the agent has followed expert instruction in the past. It is consistent with the intuition of a heuristic reward based on semantic similarity because the better the agent has been following expert instructions, the more likely it is on the right track to collect more future rewards.

Using HuRL framework allows us to analyze how false positive rewards influence on policy performance gap, defined as $V^*(s_0) - V^\pi(s_0)$. We will demonstrate that false positive rewards increase the upper bound of this gap, whereas false negative rewards maintain this upper bound.

4.3 False Positives vs. False Negatives

To begin, we provide the formal definition of false positive auxiliary rewards from both instruction-following (IF) and heuristic perspectives:

VLM reward function (Pixl2R) in our experiments.

Definition 4.4 (False Positive Rewards)**.** A false positive reward occurs when:

IF Perspective: For a trajectory τ_t that does not satisfy instruction $l_{m(t)}$, the VLM-based reward $r^v(\tau_t, l_{m(t)})$ ranges between 0 and 1.

Heuristic Perspective: The heuristic $h(s_t) > V^*(s_t)$, overestimating the optimal value of s_t.

Proposition 4.5. *False positive rewards in the IF perspective imply false positive rewards in the heuristic perspective.*

The heuristic is approximated as $h(s_t) \approx V^*(s_t) = 1 \cdot \gamma^{\widetilde{T}-t}$, calculated based on an assumed optimal path length $\widetilde{T}$. Here, $\widetilde{T} - t$ measures the remaining steps towards the goal. When a trajectory τ_t receives a high reward but fails to fulfill instruction $l_{m(t)}$, it corresponds to a high $h(s_t)$, thus a low $\widetilde{T} - t$. However, since the agent must eventually redo $l_{m(t)}$, s_t is actually further from the goal than estimated. Therefore, the actual distance $T - t$ to reach the goal will exceed $\widetilde{T} - t$, and $V^*(s_t)$, calculated with the actual T, will be smaller than $h(s_t)$, derived from $\widetilde{T}$. This thus explains how a false positive reward from an instruction-following (or VLM) perspective leads to an overestimation of the heuristic. Researchers have advocated the benefits of pessimistic value estimation to enhance the stability of RL algorithms (Kumar et al., 2020; Jin et al., 2021). In HuRL, Cheng et al. (2021) further identify a beneficial property: when a heuristic is pessimistic with respect to $\mathcal{M}$, it results in a smaller upper bound on the performance gap.

Definition 4.6 (Pessimistic h)**.** Let Bellman operator $(\mathcal{B}h)(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}[h(s')]$. A heuristic function h is said to be *pessimistic* with respect to an MDP $\mathcal{M}$ if $\max_a (\mathcal{B}h)(s, a) \geq h(s)$. This condition essentially means that the heuristic h never overestimates the true value of a state.

Proposition 4.7. *Even if the heuristic remains conservative for all successor states, a single overestimation ($h(s) > V^*(s)$) can violate the pessimistic condition by causing $max_a(\mathcal{B}h)(s, a) < h(s)$.*

Informally, our result on the impact of false positive/negative rewards on convergence is as follows:

Theorem 4.8. *Borrowing from Cheng et al. (2021), the performance gap in RL when using heuristics can be broken down into a regret term and a bias term, where false negatives maintain the upper bound of bias while false positives increase the bias by breaking the heuristic's pessimism, thereby potentially leading to slower convergence.*

See Appendix A.2.2 for formal version and detailed proofs. In the next section, we present a case study on cosine similarity metrics, demonstrating how they contribute to false positive rewards in learned reward models.

5 False Positives From Cosine Similarity

This section identifies and discusses two fundamental issues with cosine similarity scores in sequential decision-making contexts: *state entanglement* and *composition insensitivity*. The former issue, state entanglement, refers to the metric's inability to recognize trajectories 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 even when the temporal relationships between sub-tasks are not satisfied.

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 words, 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, potentially prolonging the path to the goal state by necessitating corrective actions or re-attempts.

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. We also observed this phenomenon particularly in the *Montezuma* environment, where RL agents tend to *hack* the reward system by focusing on the easiest actions that yield rewards (e.g., moving towards a direction) rather than executing more complex, timely actions. **Eventually, this leads to an overestimation of the agent's progress towards the ultimate goal.** (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. In contrast to some advancements in language models, compact visual and sentence embeddings from multimodal VLMs remain largely insensitive to sequential information (Pham et al., 2021). When the task is order-sensitive, executing actions in the wrong sequence prolongs the path towards the goal state, as agents need to re-attempt the correct order.

Figure 1 illustrates various scenarios where false positive rewards are erroneously assigned. To empirically demonstrate the issue, Section 5.1 presents experiments on these issues and their impact on agent learning in sparse reward environments.

5.1 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)** VLM reward models lacking noise handling mechanisms underperform against 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 (Hafner, 2021); (2) *Montezuma*, a classic hard adventure game in Atari (Bellemare et al., 2013); and (3) *Minigrid 'Go To Seq'*, a hard task involving long-horizon navigation and object interactions (Chevalier-Boisvert et al., 2018). A **Markovian** and a **Non-Markovian** reward model were tested: (1) *Pixl2R* by Goyal et al. (2020), 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. (2023). 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* (Radford et al., 2021), pretrained by image-text pairs; and (2) *X-CLIP* (Ma et al., 2022), 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* (Zhang et al., 2021) for comparison. It provides auxiliary rewards based on observation novelty to encourage exploration. See Appendix A.4 for detailed implementation of the experiments.

Evaluation Metric. We adopted the ***score*** metric from the Crafter benchmark (Hafner, 2021) for performance evaluation, as it effectively measures consistent performance across multiple subtasks in sparse reward environments. Unlike the *maximum total rewards* metric, which does not adequately reflect consistent performance, the score metric offers a more reliable indicator of learning progress. See Appendix A.5 for its formal definition.

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. 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, three types are evaluated: (1) matched pairs; (2) not-matched pairs and (3) manipulated pairs which are derived from matched pairs by polluting either the trajectory or the instruction. Our results reveal a critical flaw in the reward model: despite manipulated pairs that fundamentally fail to fulfill the instruction, the model paradoxically assigns high similarity scores (see Figure 3 for overall results and also Appendix A.6 for individual environments). It's worth noting that the poor performance in the negation case aligns with broader challenges in natural language processing. Recent studies (Hossain et al., 2022; Truong et al., 2023) have highlighted that negation is central to language understanding but is not properly captured by modern language models. This limitation extends to VLMs and directly leads to false positive rewards.

Prevalence of False Positives. To address **H2**, we analyzed reward distribution heatmap from VLM-

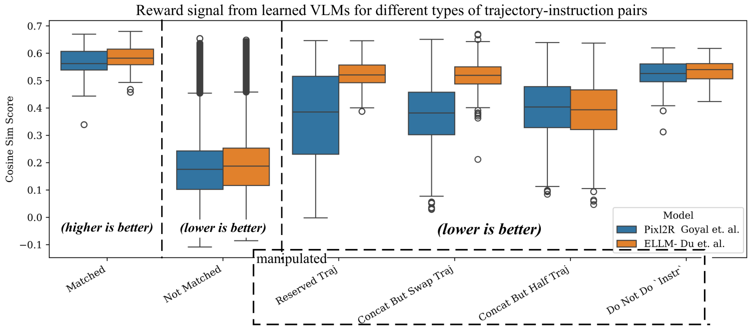


Figure 3: Learned VLM models performed badly with O.O.D. examples. They incorrectly assign high scores to manipulated pairs, which should be low as the trajectories in the manipulated pairs fail the instruction.

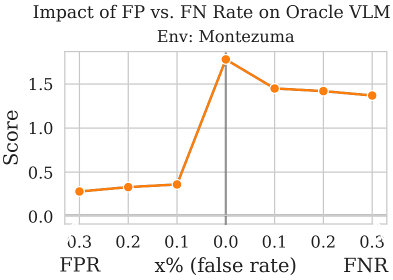


Figure 4: The false positive vs. false negative oracle model. The false positive model get a more severe drop in the final training score.

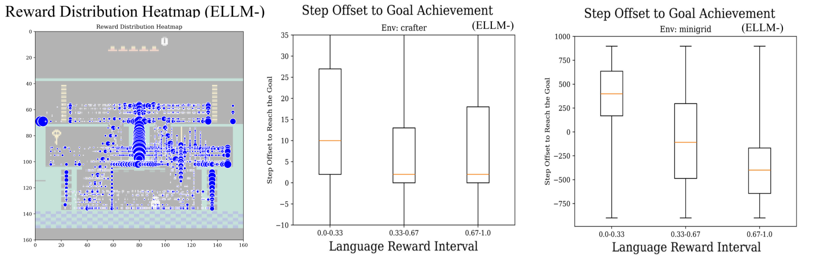


Figure 5: The heatmap shows rewards received at various locations, with larger circle sizes indicating higher rewards. The later figures shows the offsets between the state where rewards are given and the actual goal-reaching state. Agents are getting both issues of false positives and false negatives

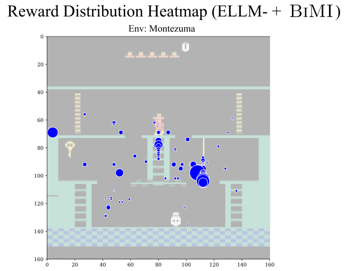


Figure 6: The ratio of false positive rewards is significantly reduced after applying BIMI

Table 1: Score metric across environments (equivalent to total rewards, higher is better). ⋆ denotes baseline intrinsic reward model. VLM reward models without noise handling underperformed. All are based on PPO.

Models	Type	Monte.	Minigrid	Crafter	% vs. DEIR
PPO	Pure	0.151	24.9	16.8	−28%
DEIR ⋆	Intrinsic	0.174	55.5	19.7	–
Pixl2R	VLM	0.142	12.4	9.40	**−49%**
ELLM-	VLM	0.150	19.4	10.8	−41%
Pixl2R + DEIR	VLM + intr.	0.176	17.3	10.4	−38%
ELLM- + DEIR	VLM + intr.	0.178	30.9	11.8	−27%

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* 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 false positive rewards. 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 a false positive reward, as the reward was given before reaching the goal. Conversely, a negative offset indicates a false negative, where the agent reached the goal but the reward model failed to acknowledge it (see Figure 5). Interestingly, besides positive offsets, we observed a large amount of negative offsets in Minigrid environments. We attribute this to Minigrid's abstract shape-based visual representations, which fall outside the VLM's training distribution.

Impact on Learning. We trained agents using learned VLM reward models and compared their learning efficacy against intrinsic reward models. As shown in Table 1, our results confirmed **H3**: *instruction-following 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 small (0.1) one-off reward for certain locations, covering x% of the map. The results indicate that false negatives were less detrimental to agent performance than false positives (see Figure 4). This performance difference can be explained through our theoretical analysis in Appendix A.2.2.

6 Addressing the False Positive Issue

Our proposed solution is not specific to cosine similarity's approximation error, which serves as a case study. Rather, it's a broad strategy for mitigating false positive rewards across various reward models, including noisy VLM reward sources.

6.1 Binary Signal and Conformal Prediction

Our experiments have demonstrated that false positives are more detrimental to learning than false negatives. Based on these findings, we propose a reward function that issues a one-time binary reward only when the similarity between the agent's current trajectory and the instruction exceeds a high confidence threshold. This approach contrasts with previous methods, which provide continuous rewards whenever the reward score exceeds a predefined threshold, and continue to do so until reaching a maximum cap. Our method, however, delivers this reward only once. This approach minimizes the likelihood of accumulating false positive rewards while maintaining adherence to Assumption 4.3.

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 thresholding approach leverages statistical properties studied by (Sadinle et al., 2019), which ensures a high probability (at least $1-\alpha$) that true positive pairs are recognized while minimizing frequency of false positives errors. See Appendix A.4.2 for the pseudocode of the instruction-following RL algorithm and Appendix A.8 for detailed threshold calculation.

6.2 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:

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

where $\tau = \langle s_{t-W}, a_{t-W}, \ldots, s_t\rangle$ is the agent's trajectory up to current time step t, and W is the memory size of the agent for its past trajectory. $p(l_k \mid \tau)$ comes from the similarity score provided by VLMs, referring to the likelihood of the instruction l_k being fulfilled by a trajectory τ. $p(l_k)$ is overall likelihood of encountering the instruction l_k in the learning environment. 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, any reward signal from this instruction will be downplayed. $p(l_k)$ is calculated as follows:

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

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. (2023). Overall, BIMI, the proposed reward function that enhances the noise resilience of VLM-based reward models, can be expressed as follows:

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

It's important to note that the BIMI approach primarily mitigates false positives (FP) rather than false negatives (FN). Both BI and MI aim to reduce the likelihood of rewarding unintended trajectories, thus addressing the FP issue. While this conservative approach may increase false negatives by

Table 2: Model score across various environments. ⋆ 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. ⋆	Minigrid	% vs. ⋆	Crafter	% vs. ⋆
Pixl2R ⋆	0.142 ± 0.003	–	12.4 ± 2.43	–	9.40 ± 0.022	–
Pixl2R + Bi	0.137 ± 0.009	−3.5%	31.2 ± 2.04	+151%	10.7 ± 0.784	+14%
Pixl2R + BiMI	0.162 ± 0.022	+14%	37.5 ± 7.83	+199%	7.95 ± 0.351	−15%
Pixl2R + DEIR	0.176 ± 0.009	+23%	17.3 ± 0.51	+39%	10.4 ± 1.015	+10%
Pixl2R + BiMI + DEIR	0.267 ± 0.016	+88%	57.7 ± 2.15	+364%	11.0 ± 0.190	+17%
ELLM- ⋆	0.150 ± 0.004	–	19.4 ± 10.06	–	10.8 ± 1.017	–
ELLM- + Bi	0.151 ± 0.016	+0.6%	29.7 ± 1.29	+53%	11.1 ± 0.601	+3.2%
ELLM- + BiMI	0.156 ± 0.014	+4.0%	33.6 ± 3.99	+74%	9.42 ± 0.267	−12%
ELLM + DEIR	0.178 ± 0.029	+20%	30.9 ± 3.50	+59%	11.8 ± 1.152	+9.5%
ELLM- + BiMI + DEIR	0.279 ± 0.078	+86%	56.2 ± 6.19	+190%	13.1 ± 0.393	+22%

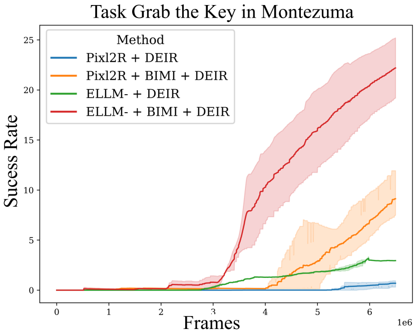


Figure 7: BIMI reward showed faster and higher success rates on difficult tasks in Montezuma

pruning some correct trajectories, this trade-off is beneficial, as demonstrated by both our empirical and theoretical results.

7 Experiments

We continue to evaluate using **Markovian** *Pixl2R* and **Non-Markovian** *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 5.1, with additional details in Appendix A.9.

7.1 Main Results

In *Montezuma*, Pixl2R+BIMI demonstrated 14% performance increase compared to the original models (see Table 2), 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.

See Figure 6 for a quantitative analysis: 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 7 demonstrates a higher success rate in grabbing the key in the first room, one of the most difficult tasks in *Montezuma*, highlighting the effectiveness of the proposed reward function and its synergy with intrinsic reward models in guiding agents to solve difficult sparse-reward tasks. See results for other environments in Appendix A.10.

7.2 Overall Performance and Ablation Study

The overall improvements were substantial. As shown in Table 2, BIMI led to a 67% improvement for *Pixl2R* (Markovian) and a 22% improvement for *ELLM-* (Non-Markovian). These results are also illustrated in Figure 16 in Appendix A.10. 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.

8 Conclusion

We reveal two findings in VLM-based reward functions for RL agents: (1) false positive rewards, rather than false negatives, are more detrimental to policy learning; and (2) our proposed BIMI reward function advocates for pessimistic rewarding, significantly mitigating the slowdown in learning caused by false positives. Our results are supported by both theoretical analysis and empirical validation across three challenging embodied tasks.

9 Limitations

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. (2024). However, we believe that finetuning large VLMs simultaneously during reinforcement learning (RL) is computationally expensive and may not be practical in real-world scenarios.

We acknowledge that mutual information (MI) has been previously explored as a reward objective in RL. However, our work uncovers a novel insight: MI is particularly well-suited for preventing over-reliance on excessively frequent reward sources, which in turn helps to mitigate the issue of too many false positive rewards. This research is innovative in explicitly addressing the practical challenges posed by false positive rewards in instruction-following learning agents that utilize VLM rewards. We believe that our proposed BiMI reward function represents a novel application of MI specifically tailored for VLM-based rewards.

There is a gap in providing a rigorous theoretical foundation for why our theoretical findings extend to non-Markovian reward models like ELLM-. Our Assumption 4.3 facilitates the integration of non-Markovian reward models into the HuRL framework, though some might argue this assumption is strong. However, with advancements in deep RL, the distinction between non-Markovian and Markovian models has become increasingly blurred. For instance, the pioneering DQN (Mnih, 2013) utilized sequences of past observations as input, ostensibly making it non-Markovian. Yet, with deep learning's capacity to encapsulate complex state histories into a representation that can be treated as Markovian at a higher abstraction level, some have argued that non-Markovian elements can be effectively reconsidered as Markovian within this new context (Hausknecht and Stone, 2015).

Our experiments were conducted in extremely sparse reward settings, with tasks intentionally designed to be challenging by requiring long-horizon planning and sequential execution. This setup may overemphasize the impact of false positive rewards. We acknowledge that in certain cases, rewarding partially correct trajectories may be beneficial for exploration. However, to prevent the agent from overvaluing these partial solutions, it is essential that the reward signal remains conservative, providing a pessimistic estimate of the value of partially correct trajectories.

References

David Abel, Will Dabney, Anna Harutyunyan, Mark K Ho, Michael Littman, Doina Precup, and Satinder Singh. 2021. On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 34:7799–7812.

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. 2021. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *J. Mach. Learn. Res.*, 22(98):1–76.

Joseph Antognini and Jascha Sohl-Dickstein. 2018. Pca of high dimensional random walks with comparison to neural network training. *Advances in Neural Information Processing Systems*, 31.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2018. Exploration by random network distillation. In *International Conference on Learning Representations*.

Yi-Chun Chen, Mykel J Kochenderfer, and Matthijs TJ Spaan. 2018. Improving offline value-function approximations for pomdps by reducing discount factors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3531–3536. IEEE.

Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. 2021. Heuristic-guided reinforcement learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 13550–13563.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*.

Jack Clark. 2016. Faulty reward functions in the wild. https://openai.com/index/faulty-reward-functions/. Accessed: 2024-08-06.

Jan Corazza et al. 2022. Reinforcement learning with stochastic reward machines. In *AAAI Conference on Artificial Intelligence*.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, P. Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*.

Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. 2024. Furl: Visual-language models as fuzzy rewards for reinforcement learning. In *Forty-first International Conference on Machine Learning*.

Gaurav R. Ghosal, Matthew Zurek, Daniel S. Brown, and Anca D. Dragan. 2022. The effect of modeling human rationality level on learning rewards from multiple feedback types. In *AAAI Conference on Artificial Intelligence*.

Prasoon Goyal et al. 2019. Using natural language for reward shaping in reinforcement learning. In *International Joint Conference on Artificial Intelligence*.

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

Danijar Hafner. 2021. Benchmarking the spectrum of agent capabilities. In *Deep RL Workshop NeurIPS 2021*.

Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

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

Md Mosharaf Hossain et al. 2022. An analysis of negation in natural language understanding corpora. *arXiv preprint arXiv:2203.08929*.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*.

Ying Jin, Zhuoran Yang, and Zhaoran Wang. 2021. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR.

Russell Kaplan et al. 2017. Beating atari with natural language guided reinforcement learning. *ArXiv*, abs/1704.05539.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191.

Mengdi Li, Xufeng Zhao, Jae Hee Lee, Cornelius Weber, and Stefan Wermter. 2023. Internally rewarded reinforcement learning. In *International Conference on Machine Learning*, pages 20556–20574. PMLR.

Yiwei Ma, Guohai Xu, Xiaoshuai Sun, Ming Yan, Ji Zhang, and Rongrong Ji. 2022. X-clip: End-to-end multi-grained contrastive learning for video-text retrieval. *Proceedings of the 30th ACM International Conference on Multimedia*.

Parsa Mahmoudieh et al. 2022. Zero-shot reward specification via grounded natural language. In *International Conference on Machine Learning*.

Volodymyr Mnih. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Seungyong Moon, Junyoung Yeom, Bumsoo Park, and Hyun Oh Song. 2023. Discovering hierarchical achievements in reinforcement learning via contrastive learning. In *Neural Information Processing Systems*.

Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.

Thang Pham, Trung Bui, Long Mai, and Anh Nguyen. 2021. 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.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.

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

Thinh Hung Truong, Timothy Baldwin, Karin Verspoor, and Trevor Cohn. 2023. Language models are not naysayers: an analysis of language models on negation benchmarks. *arXiv preprint arXiv:2306.08189*.

Xin Eric Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan fang Wang, William Yang Wang, and Lei Zhang. 2018. 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.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents. In *Neural Information Processing Systems*.

Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E. Gonzalez, and Yuandong Tian. 2021. Noveld: A simple yet effective exploration criterion. In *Neural Information Processing Systems*.

A Technical Appendix

Continued from the main text of *Understanding and Mitigating Noise in VLM Rewards*, the technical appendix consists of the following:

- **§A.1 Detailed Problem Setting**, which provides a detailed problem setting of the instruction-following reinforcement learning. This is referred to in Section 3 and Section 4.
- **§A.2 Theoretical Analysis and Proofs**, which is referred to in Section 4
- **§A.3 Additional Notes on Theoretical Setting**
- **§A.4 Implementation Details of the Experiments**, which specify the implementation details for both the first and second stage experiments in Section 5.1 and Section 7 respectively.
- **§A.5 Additional Details of the Experiments of False Positive Rewards**, which is referred in Section 5.1.
- **§A.6 Details of Showing the Prevalence of False Positives in VLM Cosine Similarity Scores**, which provides extra figures that cannot be in the main body due to page limit.
- **§A.7 Impact of Reward Model Quality on Policy Learning Efficiency: Supplementary Experiments and Quantitative Analysis in Montezuma**, which provides extra evaluation results to further illustrate false positive rewards hinders agent learning.
- **§A.8 Pseudo-code for Empirical Quantile Calculation for Binary Signal Threshold**, which is referred in Section 6.1.
- **§A.9 Additional Implementation Details of the Experiments of BIMI Reward Function**, which is referred in Section 7.
- **§A.10 Detailed Experiment Results of BIMI Reward Function**, which provides detailed experiment results of BIMI reward function, which is referred in Section 7.
- **§A.11 Deprecated Convergence Analysis**, It is retained here solely for reference and backtracking purposes. As the more rigorous theoretical analysis is constructed, the content of this section is now **deprecated**.

A.1 Detailed Problem Setting

A.1.1 The Pointer Mechanism in Reward Machine

In our problem setting, $r^v(\tau_t, l_{m(t)})$ is the VLM based reward at time step t evaluated using the sub-trajectory τ_t is the sub-trajectory up to time t.

Existing VLM-reward implementations applied a pointer mechanism that decide which instruction should the agent consider at the current step. We follow this implementation and denote the pointer as $m(t)$. It indicates the current instruction of the agent is trying to complete at time step t. $m(t)$ is updated according to the following rule:

$$m(t+1) = \begin{cases} 1 & \text{if } t = 0 \\ m(t) + 1 & \text{if instr. } l_{m(t)} \text{ completed at } t \\ m(t) & \text{otherwise} \end{cases} \tag{2}$$

In both the ELLM and the Pixl2R framework that uses VLM-based rewards, the assessment of whether instruction $l_{m(t)}$ is completed at time t is based on the cumulative reward for that instruction. The pointer remains on the current instruction until the accumulated reward reaches a predetermined threshold. For a detailed pseudo-code of the existing VLM reward + RL algorithm, please refer to Appendix A.4.2. Note that our proposed BIMI reward function uses a different mechanism to determine if instruction $l_{m(t)}$ is completed at time t.

In logic, **an atomic sentence** is a type of statement which cannot be broken down into other simpler sentences. While the pointer mechanism effectively enforces the sequential ordering of instructions at a higher level, each individual instruction within the expert walkthrough is not necessarily **atomic**. This means that a single instruction can encapsulate multiple finer-grained requirements for the agent's behavior. Therefore, even though the pointer mechanism is implemented, the internal composition of each instruction is not strictly enforced. Consequently, the issue of "composition insensitivity" can arise when the VLM tries to align agents' trajectories with non-atomic instruction sentences.

Just as potential-based reward shaping (PBRS) modifies the original MDP into a reshaped MDP through the use of auxiliary rewards, the HuRL framework similarly transforms the original MDP $\mathcal{M}$ into a new MDP $\widetilde{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_0, \widetilde{r}, \widetilde{\gamma} \rangle$, employing a heuristic-based auxiliary reward h, along with a coefficient $\beta \in [0, 1]$ that scales the auxiliary rewards. Thus, the updated reward function can be written as:

$$\widetilde{r}(s, a) = r^e(s, a, s') + (1 - \beta)\gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}[h(s')] \quad \text{and } \widetilde{\gamma} = \beta\gamma$$

We also provide the formal definition of false negative reward as below.

Definition A.1 (False Negative Rewards)**.** A false negative reward occurs when:

Instruction-Following Perspective: The VLM-based reward $r^v(\tau_t, l_{m(t)}) \approx 0$ for a trajectory τ_t that *does* satisfy instruction $l_{m(t)}$.

Heuristic Perspective: The heuristic $h(s_t) \approx 0 < V^*(s_t)$, underestimating the optimal value of s_t.

A.1.2 Detailed Terminology

This section contains detailed terminology from both the standard RL and the HuRL framework.

We start by setting clear the terminology. The original MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_0, r^e, \gamma \rangle$ as set out in the main text. Let $G^e(\tau)$ denote the cumulative environment reward of a single trajectory $\tau = \langle s_0, a_0, \dots, s_T \rangle$ where T is the total time of the trajectory, we have

$$G^e(\tau) = \sum_{t=0}^{T} \gamma^t r^e(s_t, a_t). \tag{3}$$

A policy π_θ is a conditional distribution $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ parameterized by θ, where $\Delta(\cdot)$ denotes the space of probability distribution. The probability of a trajectory τ being generated, given that actions come from π_θ and the starting state is s_0, is expressed as:

$$P(\tau \mid \pi_\theta) = \prod_{t=0}^{T} \mathcal{P}(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t) \tag{4}$$

Because a policy π is stochastic, different trajectories could be generated from the same policy for a given initial state. The objective function to be optimized is the expected cumulative environment reward of trajectories generated by π starting from state s_0. It is also called the state value function of s_0 and denoted as:

$$V^\pi(s_0) := \mathbb{E}_{\rho_s^\pi}\left[G^e(\tau)\right] = \sum_{\text{all } \tau_i \text{ generated by } \pi_\theta} G^e(\tau_i) \cdot P(\tau_i|\pi) \tag{5}$$

where ρ_s^π denotes the trajectory **distribution** of $s_0, a_0, s_1, ...$ induced by running π starting from $s_0 = s$. We defined an acceptable policy π_A as

$$\pi_A : V^{\pi_A} \geq V^{\pi^*} - \epsilon \tag{6}$$

where π^* is the optimal policy and ϵ is a control parameter that quantifies the acceptance of other policies. Acceptable policies defined in this way are also referred to as ϵ-optimal policies. We set $V^* = V^{\pi^*}$ for easy reference. Any policy that is not an acceptable policy is an unacceptable policy π_U. We define a good trajectory τ_G as

$$\tau_G : G^e(\tau_G) \geq V^* - \epsilon. \tag{7}$$

Any trajectory that is not a good trajectory is a bad trajectory τ_B. Note that this way of distinguishing good/bad trajectories and acceptable/unacceptable policies is explicitly based on the environment reward r^e, and introducing additional or alternative rewards into the system does not change this definition. For easy reference, we call any trajectory that reaches a goal state a goal trajectory, and we call any policy that can generate at least one goal trajectory a goal policy.

We denote the state distribution of a policy π at time t as d_t^π. Thus, the **discounted average state distribution** of a policy π can be expressed as $d^\pi = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t d_t^\pi$, where $(1-\gamma)$ is the normalization factor to ensure the result is a proper probability distribution. We also define a shorthand notation – for a state distribution $d \in \Delta(\mathcal{S})$, we define $V(d) = \mathbb{E}_{s\sim d}[V(s)]$.

When the Heuristic-Guided Reinforcement Learning (HuRL) framework is applied, the reshaped MDP $\widetilde{\mathcal{M}}$ has separated reshaped reward $\widetilde{r}$ and separated discount factor $\widetilde{\gamma} = \beta\gamma$, where $\beta \in [0, 1]$ is the hyperparameter that scales the auxiliary heuristic rewards. We denote the value function of the reshaped MDP as $\widetilde{V}$, and the optimal policy and its value function under $\widetilde{\mathcal{M}}$ as $\widetilde{\pi}^*$ and $\widetilde{V}^*(\widetilde{V}^{\widetilde{\pi}^*})$.

The discussion regarding the convergence guarantee of learning π in $\widetilde{\mathcal{M}}$ resulting in the optimal policy in the original MDP $\mathcal{M}$ can be found in Appendix A.3.1. For the purposes of this paper, it is sufficient to note that the authors of HuRL have provided a trick to ensure convergence. Moreover, empirical experiments conducted by the authors demonstrate that HuRL converges even without this trick.

It is also important to distinguish the Bellman backup equation under the two different MDPs: the original $\mathcal{M}$ and the reshaped $\widetilde{\mathcal{M}}$. By definition, $(\mathcal{B}h)(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')]$. In contrast, $(\widetilde{\mathcal{B}}h)(s,a) = \widetilde{r}(s,a) + \widetilde{\gamma}\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')]$. Nevertheless, the two Bellman backup equations possess a remarkable property – they are essentially equivalent to each other:

$$\begin{aligned}
(\widetilde{\mathcal{B}}h)(s,a) &= \widetilde{r}(s,a) + \widetilde{\gamma}\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')] \\
&= \big(r(s,a) + (1-\beta)\gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')]\big) + \beta\gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')] \\
&= r(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[h(s')] \\
&= (\mathcal{B}h)(s,a)
\end{aligned} \tag{8}$$

A.1.3 Detailed Construction of Markovian Heuristic using Non-Markovian Rewards from VLM

Specifically,

$$h(s_t) := \begin{cases} A(s_t)V^*(s_t) & \text{if } s_t \in S^{m(t)} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where

$$A(s_t) := \frac{\Pi_{t'=0}^{t} r^v(\tau_{t'}, l_{m(t')})}{\Pi_{t'=0}^{t} r^v(\tau_{t'}^*, l_{m(t')})} \in [0, 1] \tag{10}$$

is a scaling factor that quantifies how closely the agent is following previous instructions.

A.2 Theoretical Analysis and Proofs

A.2.1 Convergence time on a sparse-reward landscape

We can pin down a characteristic convergence time on a sparse-reward landscape. The sparse-reward setting enforces that

$$r^e(s_t) = \begin{cases} 1 & \text{if } s_t \in \mathcal{S}_G \\ 0 & \text{otherwise} \end{cases}. \tag{11}$$

It is clear from the definition that a good trajectory must always be a goal trajectory, and the cumulative reward is simply the reward at the final goal state so $G^e(\tau) = \gamma^T$ in this setting. For a randomly initialized policy, it is highly unlikely that the initial distribution of trajectories contains any goal trajectory due to the sparsity of goal states. The optimization of $V^e_{\pi_\theta}$ thus consists of two parts. The first part is to search for a goal trajectory. The gradient landscape is almost 0 everywhere, except for cases where a trajectory is δ-close to a goal trajectory. Here δ is the differential unit in the numerical differentiation used in the gradient calculation

$$\theta = \theta + \alpha \nabla_\theta V_{\pi_\theta} \tag{12}$$

such that δ-close means being numerically accessible within a distance of $|\delta|$ in parameter space. And the second part is to reduce T so that goal trajectories become good trajectories and consequently achieving acceptable policies. For the first part, searching for a trajectory for the target is effectively a random walk in the d-dimensional parameter space due to the flat gradient landscape.

Lemma A.2. *For a random walk in n-dimensional space, the expected number of steps T_D needed to travel a distance of D scales with D^2.*

Proof: let $\vec{X}_1$, $\vec{X}_2$, ... $\vec{X}_T$ be IID random unit vectors uniformly distributed on a $(d-1)$-dimensional sphere $\mathcal{S}^{d-1} \subset \mathbb{R}^d$, where $\vec{X}_i = (X_{i1}, ..., X_{id})$ and $|\vec{X}_i|^2 = \sum_{j=1}^d X_{ij}^2 = 1$. Let $\vec{S}_T := \sum_{i=1}^T \vec{X}_i$. By a n-dimensional cosine rule we have

$$|\vec{S}_T|^2 = |\vec{S}_{T-1}|^2 + 2\vec{S}_{T-1} \cdot \vec{X}_T + |\vec{X}_T|^2, \tag{13}$$

and because $\mathbb{E}[\vec{X}_T] = \vec{0}$

$$\begin{aligned}
\mathbb{E}[|\vec{S}_T|^2] &= \mathbb{E}[|\vec{S}_{T-1}|^2] + \mathbb{E}[2\vec{S}_{T-1} \cdot \vec{X}_T] + 1 && (14)\\
&= \mathbb{E}[|\vec{S}_{T-1}|^2)] + 2\vec{S}_{T-1}\mathbb{E}[\vec{X}_T] + 1 && (15)\\
&= \mathbb{E}[|\vec{S}_{T-1}|^2] + 1 && (16)\\
&= \mathbb{E}[|\vec{S}_{T-2}|^2] + 1 + 1 && (17)\\
&\vdots && (18)\\
&= T && (19)
\end{aligned}$$

i.e. $\mathbb{E}[|\vec{S}_T|] \sim \sqrt{T}$. When a policy is randomly initialized with θ_0, the distance D to a goal policy, $D := ||\theta_{goal} - \theta_0||$ is fixed and is the distance the random walk needs to travel ($\mathbb{E}[|\vec{S}_T|] = D \sim \sqrt{T_D}$), so the characteristic time needed to travel this distance $T_D \sim D^2$ as we have shown above.

Assumption 4.1. Expert knowledge can guide the parameter search along a path in parameter space, defined by a sequence of n intermediate parameter vectors $\theta_1, \ldots, \theta_n$, where each θ_i represents the parameters after learning sub-task l_i. As a result, the overall distance D can be decomposed into segments: $D \approx \sum_{k=1}^{n-1} d_i$, where $d_i = ||\theta_{i+1} - \theta_i||$.

Auxiliary rewards essentially open the path for a divide-and-conquer approach by introducing intermediate rewards in the learning process. We introduce BiMI rewards as

$$r^v_{\text{BiMI}}(\tau, l_k) = \max(\mathbf{1}_{\{p(l_k|\tau) \geq \hat{q}\}} - p(l_k), 0) \tag{20}$$

and the cumulative reward of a single trajectory in the presence of BiMI rewards becomes

$$G^v_{\text{BiMI}}(\tau) = \sum_{t=0}^{T} \gamma^t r^v_{\text{BiMI}}(\tau, l_{m(t)}). \tag{21}$$

Because r^v_{BiMI} is either $1-p(l_k)$ or 0, it effectively breaks the entire task into n segments of sub-tasks $\{l_1, l_2, ..., l_n\}$ and each sub-task is a sparse-reward problem. Because this decomposition is based on expert knowledge, we can reasonably assume that the start-finish distance D in parameter space is partitioned into $D \approx d_1 + d_2 + ... + d_{n-1}$ without incurring much detour (Assumption 4.1.)

Proposition 4.2. *The sum of expected time for a series of random walks, each covering the shorter distance of an individual sub-task, is less than the expected time to travel the entire distance D in one long random walk:*

$$\frac{1}{n-1}\mathbb{E}[T_D] \leq \mathbb{E}\left[\sum_{i=1}^{n-1} T_{d_i}\right] < \mathbb{E}[T_D].$$

Proof: the expected time taken for each of the sub-tasks then scales with d_i^2 respectively (Lemma A.2), and we have

$$d_1^2 + d_2^2 + ... + d_{n-1}^2 < (d_1 + d_2 + ... + d_{n-1})^2 \tag{22}$$

$$\mathbb{E}\left[\sum_{i=1}^{n-1} T_{d_i}\right] < \mathbb{E}[T_D] \tag{23}$$

because $d_i > 0 \forall i$. We can also work out that the upper bound for this improvement is a factor of $n-1$ by invoking the Cauchy–Schwarz inequality $\left(\sum_{i=1}^n u_i^2\right)\left(\sum_{i=1}^n v_i^2\right) \geq \left(\sum_{i=1}^n u_i v_i\right)^2$:

$$(d_1^2 + d_2^2 + ... + d_{n-1}^2)(1^2 + 1^2 + ... + 1^2) \geq (d_1 + d_2 + ... + d_{n-1})^2 \tag{24}$$

$$(d_1^2 + d_2^2 + ... + d_{n-1}^2) \geq \frac{1}{n-1} D^2 \tag{25}$$

$$\mathbb{E}\left[\sum_{i=1}^{n-1} T_{d_i}\right] \geq \frac{1}{n-1}\mathbb{E}[T_D] \tag{26}$$

and the equality sign holds (indicating maximal improvement) when $d_1 = d_2 = ... = d_{n-1}$. The intuitive interpretation is that the divide-and-conquer approach is the most effective when the task is divided evenly into subtasks. Additionally, the presence of $p(l_k)$ introduces some lexical-level tolerance in the learning process and results in a considerably larger δ-close radius which could result in further improvement in the convergence rate. However, as we will discuss in the next section, introducing too much tolerance could harm the convergence rate rather than improve it.

A.2.2 False Positive and the Violation of Pessimistic Property of Heuristics

Proposition 4.7. *Even if the heuristic remains conservative for all successor states, a single overestimation ($h(s) > V^*(s)$) can violate the pessimistic condition by causing $max_a(\mathcal{B}h)(s,a) < h(s)$.*

Proof. We want to show that given:

1. for all successor states s': $h(s') \leq V^*(s')$ for an arbitrary s.
2. false positive at current arbitrary state s: $V^*(s) < h(s)$.

The objective is to show that under the above conditions, we obtain $\max_a(\mathcal{B}h)(s,a) < h(s)$.

1. **Express the Bellman backup for** h: $(\mathcal{B}h)(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)h(s')$
2. **Express the Bellman version of** V^*: $V^*(s) = \max_{a\in\mathcal{A}} [R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')]$.

   Given that $h(s') \leq V^*(s')$ for all s', we can expand it to $\sum_{s'} \mathcal{P}(s'|s,a)h(s') \leq \sum_{s'} \mathcal{P}(s'|s,a)V^*(s')$. Therefore, we have:

$$R(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)h(s') \leq R(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)V^*(s') = Q^*(s,a) \tag{27}$$

3. **Taking the maximum over actions to both sides**, we have:

$$\max_a \left(R(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|s,a)h(s') \right) \leq max_a Q^*(s,a) \tag{28}$$

$$\max_a (\mathcal{B}h)(s,a) \leq V^*(s) \tag{29}$$

4. **Apply false positive condition**: Given $V^*(s) < h(s)$, substituting into the above inequality, we have $\max_a(\mathcal{B}h)(s,a) < h(s)$

Implication: Maintaining a pessimistic heuristic is inherently fragile because the introduction of a false positive in any state disrupts the pessimistic condition. □

Theorem 4.8. *Borrowing from Cheng et al. (2021), the performance gap in RL when using heuristics can be broken down into a regret term and a bias term, where false negatives maintain the upper bound of bias while false positives increase the bias by breaking the heuristic's pessimism, thereby potentially leading to slower convergence.*

We analyze the convergence speed of policy learning by examining the performance gap, which is defined as the difference between the optimal value of the initial state s_0, $V^*(s_0)$, and the value of the initial state under an arbitrary policy π, $V^\pi(s_0)$. Specifically, we focus on deriving an upper bound for this performance gap. The key intuition is that a smaller upper bound implies faster convergence to the optimal policy, as fewer iterations of policy updates will be required to reach the optimum.

We begin by stating the theorem made by HuRL authors:

Theorem A.3 (Performance Gap Decomposition (Cheng et al., 2021))**.** *For any policy π, heuristic $h : \mathcal{S} \to \mathbb{R}$, and mixing coefficient $\beta \in [0, 1]$,*

$$V^*(s_0) - V^\pi(s_0) = \text{Regret}(h, \beta, \pi) + \text{Bias}(h, \beta, \pi) \tag{30}$$

where the regret and the bias term are expressed as follows:

$$\text{Regret}(h, \beta, \pi) := \beta \left(\widetilde{V}^*(s_0) - \widetilde{V}^\pi(s_0) \right) + \frac{1-\beta}{1-\gamma} \left(\widetilde{V}^*(d^\pi) - \widetilde{V}^\pi(d^\pi) \right) \tag{31}$$

$$\text{Bias}(h, \beta, \pi) := \left(V^*(s_0) - \widetilde{V}^*(s_0) \right) + \frac{\gamma(1-\beta)}{1-\gamma} \mathbb{E}_{s,a\sim d^\pi} \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \left[h(s') - \widetilde{V}^*(s') \right] \tag{32}$$

We will not provide a proof for Theorem A.3 in this paper. Please refer to (Cheng et al., 2021) for details. The theorem demonstrates that the performance gap can be elegantly decomposed into two components: a regret term and a bias term such that:

1. The regret term quantifies the difference between $\widetilde{V^*}$ and $\widetilde{V^\pi}$, representing the error caused by π being suboptimal in the reshaped MDP $\widetilde{\mathcal{M}}$. Since π is trained by our selected RL algorithm directly on the reshaped MDP $\widetilde{\mathcal{M}}$, the primary responsibility for minimizing this regret term falls to the RL algorithm itself, not to the design of the auxiliary reward signal. Thus, when evaluating the effects of false positive or false negative rewards, we choose not to focus on bounding the regret term.

2. The bias term captures two key discrepancies: first, between the true optimal value function V^* of the original MDP $\mathcal{M}$ and the optimal value function $\widetilde{V}^*$ of the reshaped MDP $\widetilde{\mathcal{M}}$; second, between $\widetilde{V}^*$ and the heuristic $h(s)$. This term, therefore, reflects the error introduced by addressing the reshaped MDP instead of the original one, alongside how well the heuristic h approximates the optimal value function in the reshaped MDP. Consequently, this bias term directly relates to the quality of the heuristic reward signal h. Hence, we focus on analyzing its upper bound to assess the impact of false positive or false negative rewards on this heuristic.

Cheng et al. (2021) have further derived the upper bound for the bias term, which we present here as a lemma. We omit the proof for brevity; for detailed proof, please refer to (Cheng et al., 2021).

Lemma A.4 (Upper Bound of the Bias Term)**.**

$$\mathrm{Bias}(h, \beta, \pi) \leq (1-\beta)\gamma \left(\mathbb{E}_{\rho^{\pi^*}} \left[\sum_{t=1}^{\infty} (\beta\gamma)^{t-1} (V^*(s_t) - h(s_t)) \right] + \mathbb{E}_{\rho^{\pi}} \left[\sum_{t=1}^{\infty} \gamma^{t-1} (h(s_t) - \widetilde{V}^*(s_t)) \right] \right) \tag{33}$$

where ρ^{π} denotes the trajectory **distribution** of $s_0, a_0, s_1, ...$ induced by running π starting from s_0.

This upper bound elegantly illustrates a trade-off: if the heuristic h is set too high (overestimation error), it reduces the first term but increases the second term. Conversely, if h is set too low (underestimation error), it increases the first term but reduces the second term. Just by inspection, it is not immediately clear whether overestimation or underestimation results in a better upper bound, as both impact the bias term in opposing ways.

We now prove that underestimation of h (pessimistic h) is better than overestimation (i.e., $\exists s_t \in \mathcal{S}, h(s_t) > V^*(s_t)$) for minimizing bias.

Proof. 1. **Define Terms**:

- Let $B_1 = \mathbb{E}_{\rho^{\pi^*}} \left[\sum_{t=1}^{\infty} (\beta\gamma)^{t-1} (V^*(s_t) - h(s_t)) \right]$
- Let $B_2 = \mathbb{E}_{\rho^{\pi}} \left[\sum_{t=1}^{\infty} \gamma^{t-1} (h(s_t) - \widetilde{V}^*(s_t)) \right]$

2. **Underestimation Case (Pessimistic** h**)**:

Lemma A.5. *If h is pessimistic with respect to $\mathcal{M}$, $\forall \beta \in [0,1], s \in \mathcal{S}, \widetilde{V}^*(s) \geq h(s)$*

Proof. To begin with, we need to use another lemma from Cheng et al. (2021), shown as follows:

Lemma A.6 (Bellman backup of reshaped MDP (Cheng et al., 2021))**.** *For any policy π, we have*

$$\widetilde{V}^{\pi}(s_0) - h(s_0) = \frac{1}{1-\beta\gamma} \mathbb{E}_{\widetilde{d}^{\pi}_{s_0}} [(\widetilde{\mathcal{B}}h)(s,a) - h(s)] \tag{34}$$

*where $\widetilde{d}^{\pi}_{s_0}$ refers to the **discounted average state distribution** of policy π in reshaped MDP $\widetilde{\mathcal{M}}$. It can be expressed as $\widetilde{d}^{\pi}_{s_0} = (1-\widetilde{\gamma}) \sum_{t=0}^{\infty} \widetilde{\gamma}^t d^{\pi}_t$, and d^{π}_t is the state distribution of policy π at time t with $d^{\pi}_0 = \mathbf{1}\{s = s_0\}$.*

For brevity, we do not include the proof for Lemma A.6.

First of all, due to the definition of optimal value V^*, we have

$$\widetilde{V}^*(s_0) \geq \widetilde{V}^{\pi}(s_0) \tag{35}$$

Then, according to Lemma A.6, we can get

$$\widetilde{V}^*(s_0) \geq \widetilde{V}^{\pi}(s_0) = h(s_0) + \frac{1}{1-\beta\gamma} \mathbb{E}_{\widetilde{d}^{\pi}_{s_0}} [(\widetilde{\mathcal{B}}h)(s,a) - h(s)] \tag{36}$$

Let π denote the greedy policy of $\arg\max_a (\mathcal{B}h)(s,a)$ and then trace back to Equation 8, we have $(\widetilde{\mathcal{B}}h)(s,a) = (\mathcal{B}h)(s,a)$, that means

$$\widetilde{V}^*(s_0) \geq \widetilde{V}^{\pi}(s_0) = h(s_0) + \frac{1}{1-\beta\gamma} \mathbb{E}_{\widetilde{d}^{\pi}_{s_0}} [(\widetilde{\mathcal{B}}h)(s,a) - h(s)] \tag{37}$$

$$= h(s_0) + \frac{1}{1-\beta\gamma} \mathbb{E}_{\widetilde{d}^{\pi}_{s_0}} [(\mathcal{B}h)(s,a) - h(s)] \tag{38}$$

$$\geq h(s_0) \quad \text{[direct result of } h \text{ being pessimistic]} \tag{39}$$

Generalization to Any State: The lemma from Cheng et al. (2021) (Lemma A.6) is stated in terms of any starting state s_0. Therefore, we can replace s_0 with any state $s \in \mathcal{S}$ in our analysis. Thus we get $\forall s \in \mathcal{S}, \widetilde{V}^*(s) \geq h(s)$ □

Lemma A.5 implies that when h is pessimistic, $B_2 = \mathbb{E}_{\rho^\pi}\left[\sum_{t=1}^{\infty}\gamma^{t-1}(h(s_t) - \widetilde{V}^*(s_t))\right] \leq 0$ because $\widetilde{V}^*(s) \geq h(s)$ for all s. Therefore, the bias error is only bounded by B_1 when h is pessimistic, i.e., $\mathrm{Bias}(h, \beta, \pi) \leq B_1$.

3. **Overestimation Case ($\exists s_t \in \mathcal{S}, h(s_t) > V^*(s_t)$):**

   We show briefly that, unlike the underestimation error, the overestimation error does not have a closed-form upper bound expression. The difficulty, as pointed out by (Cheng et al., 2021), originates from the trajectory-dependence on $\mathbb{E}_{\rho^\pi}[\cdot]$ as the l_∞ approximation error here can be difficult to control in large state spaces. It is possible that, upon picking up falsely high h states, ρ^π gets further distorted away from ρ^{π^*} and accumulates more falsely high h states. In other words, this trajectory dependence makes B_2 prone to a feedback loop of accumulating overestimation errors and resulting in a much larger upper bound for the bias.

□

A.3 Additional Notes on Theoretical Setting

A.3.1 Convergence Guarantee of HuRL Compared With Potential-Based Reward Shaping

Previous work on potential-based reward shaping (PBRS) has established that learning a policy on a reshaped MDP can converge to the optimum of the original MDP, as proved by Ng et al. (1999). They proved that the optimal Q value of the reshaped MDP is equivalent to the optimal Q value of the original MDP minus a state-dependent function. Consequently, for an optimal policy defined as $\pi^* = \arg\max_{a\in\mathcal{A}} Q^*(s, a)$, this additional state-dependent value does not alter the policy. Thus, the optimal policy of the original MDP $\mathcal{M}$ is also the optimal policy for the reshaped $\widetilde{\mathcal{M}}$. However, the theoretical convergence for the reshaped MDP within the framework of Heuristic-Guided Reinforcement Learning (HuRL) has not been proven with the same rigor.

The authors of HuRL made a trick which involves manipulating the coefficient β, which scales the heuristic-based auxiliary rewards. If β increases gradually from 0 to 1 over the training process, the agent effectively transitions from interacting with the modified MDP back to the original MDP. This method ensures that, at the end of training, the agent is exactly solving the original problem. Moreover, according to the Blackwell optimal property (Chen et al., 2018), convergence can occur before β reaches 1, thereby allowing the trained policy to maintain optimality in the original MDP under HuRL.

However, in the original HuRL paper, among the five test environments, the β hyperparameter was actually fixed for four of them. Surprisingly, HuRL still converged to the optimal policy faster than the original MDP without updating β. This observation poses an enigma within the HuRL framework, as it is unclear why this convergence to the optimum of the original MDP occurs without dynamically adjusting β.

Thus, while the practical success of HuRL in these environments is evident, the underlying reasons for such convergence remain to be fully understood. This discrepancy between theoretical expectation and empirical results leaves room for future research to explore why HuRL converges under fixed β. Nevertheless, proving the convergence properties goes beyond the scope of our paper, which aims to highlight the prevalence of false positive rewards and their impact. An alternative perspective on this issue is to consider the learning process of maximizing auxiliary rewards as akin to performing behavior cloning from experts. In this sense, we do not need to focus on convergence within the current MDP but can view it as a form of pretraining.

A.4 Implementation Details of the Experiments

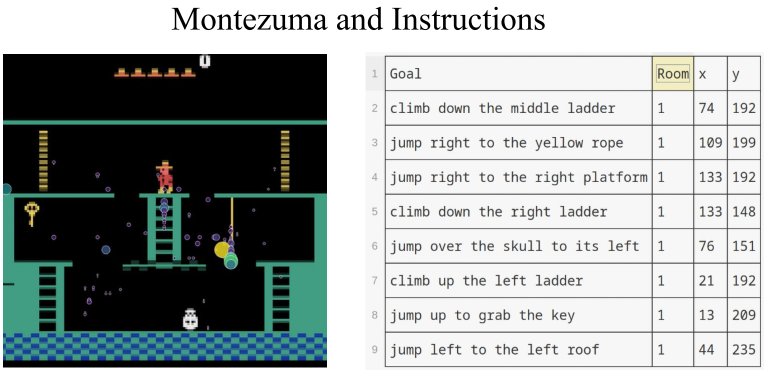
A.4.1 Environment Details

We describe each testing environment used in our experiments. More details introduction can be found in on the official project homepage of each benchmark (Chevalier-Boisvert et al., 2018; Bellemare et al., 2013; Hafner, 2021).

- **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.

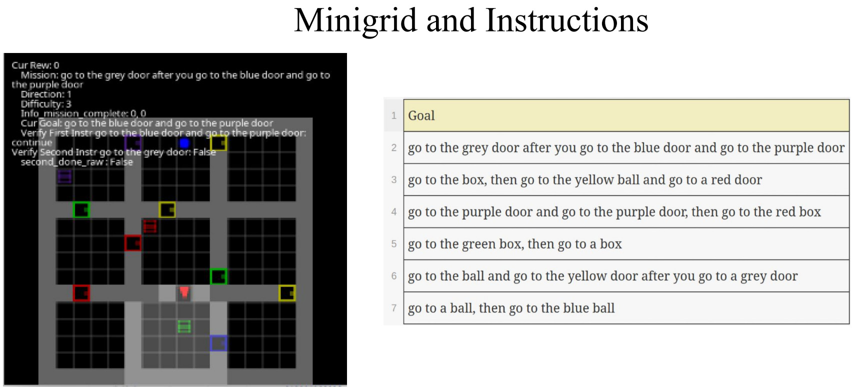
- **Minigrid 'Go to seq' Task**: We use the 'Go to seq' task in the Minigrid environment, where the agent must navigate through a sequence of rooms and touch target objects in the correct order. This is a sparse reward task where the agent receives a reward of 1 only upon completing the entire sequence 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 8: Illustration of the Montezuma's Revenge task. The agent must navigate through a series of rooms to collect treasures and keys.

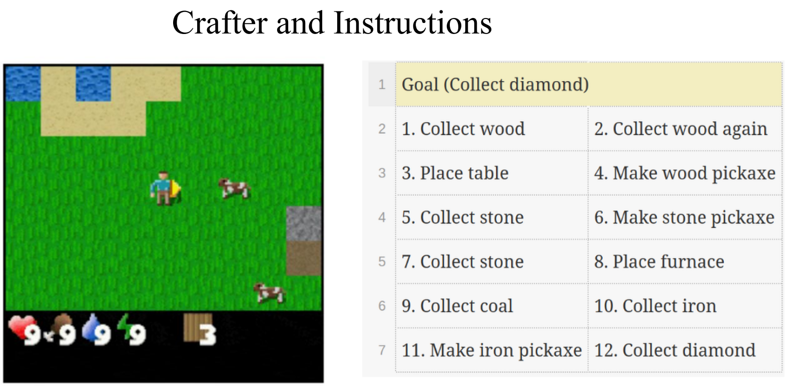


Minigrid and Instructions

```
Cur Rew: 0
   Mission: go to the grey door after you go to the blue door and go to
the purple door
   Direction: 1
   Difficulty: 3
   Info_mission_complete: 0, 0
   Cur Goal: go to the blue door and go to the purple door
   Verify First Instr go to the blue door and go to the purple door:
continue
Verify Second Instr go to the grey door: False
   second_done_raw : False
```

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 9: 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 10: Illustration of the Crafter task. The agent must survive as long as possible and explore for new crafting recipes.

A.4.2 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 (yes, the full instruction essay will be split into multiple sentences and treat each sentence as atomic instruction l_k), 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. Below is the pseudo-code for the instruction-following RL training procedure for both *Pixl2R* and *ELLM-* models.

A.4.3 Finetuning VLM-based Reward Models

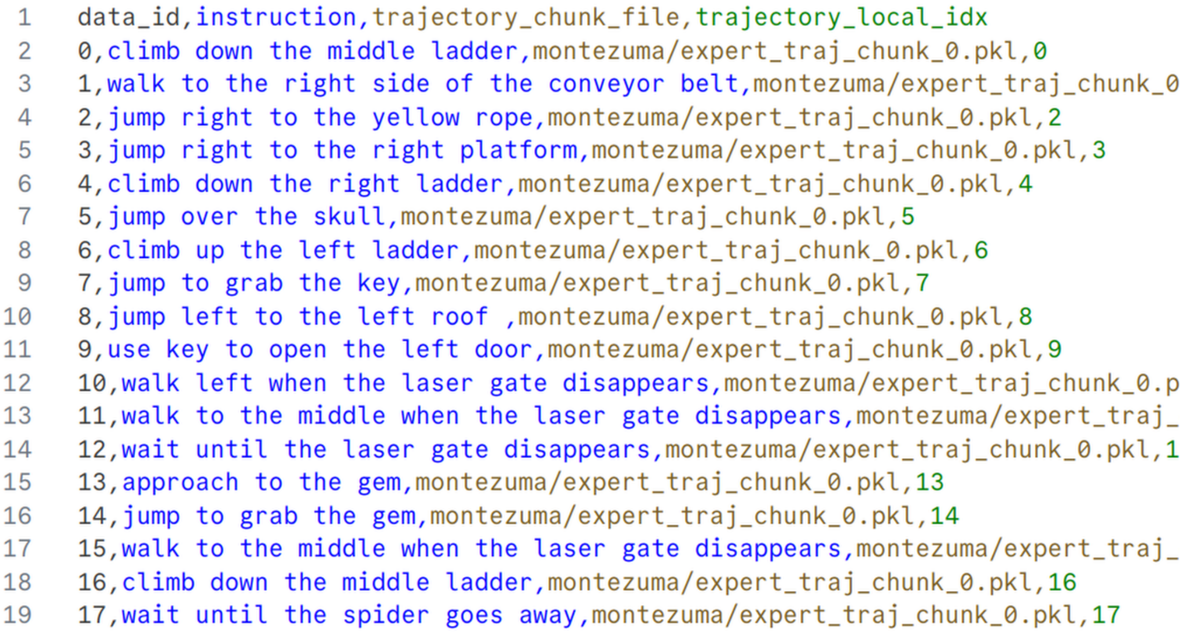
In contrast to previous work on instructing following RL where they rely on hand-crafted oracle multimodal reward models, we use actual pretrained VLMs to generate reward signals. 2 VLM backbone models are used in our experiments: 1) *CLIP* (Radford et al., 2021), pretrained by image-text pairs; and (2) *X-CLIP* (Ma et al., 2022), 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. (2023) 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.

Algorithm 1 ELLM and Pixl2R pseudo-code for instruction-following RL

```
Initialize policy network $\pi_\theta$
Initialize value network $V_\phi$
Setup VLM-based reward model $E(\cdot)$
Split instruction essay into sentences $\{l_1, l_2, ..., l_K\}$
Initialize instruction pointer $p = 1$
Initialize cumulative VLM reward $r_{\text{cum}} = 0$
Initialize cumulative VLM reward threshold $q$
Initialize replay buffer $\mathcal{D}$
Initialize agent trajectory memory queue $\tau$ with length $W$
for each episode do
    Initialize state $s_0$
    for $t = 0$ to $T - 1$ do
        Select action $a_t \sim \pi_\theta(a_t|s_t)$
        Execute $a_t$, observe next state $s_{t+1}$ and extrinsic reward $r_t^e$
        Enqueue $(s_t, a_t, r_t, s_{t+1})$ in $\tau$
        Compute VLM reward:
        $r_t^v = \frac{E(\tau) \cdot E(l_p)}{\|E(\tau)\| \|E(l_p)\|}$          ▷ Apply BiMI here if needed
        Combine rewards: $r_t = r_t^e + (1 - \beta)\gamma r_t^v$          ▷ $\beta$ is a scaling factor
        Store $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
        $r_{\text{cum}} \leftarrow r_{\text{cum}} + r_t^v$
        if $r_{\text{cum}} \geq q$ then
            $p \leftarrow \min(p + 1, K)$
            $r_{\text{cum}} \leftarrow 0$
        if Reach Update Frequency then
            Sample mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}$ from $\mathcal{D}$
            Compute TD errors:
            $\delta_j = r_j + \gamma V_\phi(s_{j+1}) - V_\phi(s_j)$
            Update value network:
            $\phi \leftarrow \phi + \alpha_v \sum_j \delta_j \nabla_\phi V_\phi(s_j)$
            Update policy network:
            $\theta \leftarrow \theta + \alpha_p \sum_j \delta_j \nabla_\theta \log \pi_\theta(a_j|s_j)$
```



```
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 11: Example of training data for the Montezuma environment.

We used the threshold $\hat{q}$ introduced in Section 6.1 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 3. **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 3: 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.4.4 Hyperparameters for Instruction-Following 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 (Moon et al., 2023). 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 4: 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 5: 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 6: 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.5 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. Formally, the score metric is defined as follows:

$$\text{Score} = \exp\left(\frac{1}{N}\sum_{k=1}^{N}\ln(1+s_k)\right) - 1 \tag{40}$$

where s_k is the agent's success rate of achieving instruction l_k, and N is the total number of instructions.

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.5.1 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.6 Details of Showing the Prevalence of False Positives in VLM Cosine Similarity Scores

We list the figures of reward signals from learned VLMs for different types of trajectory-instruction pairs for each individual environment. All figures show that the VLM-based reward models assign high rewards to manipulated trajectory-instruction pairs, indicating the prevalence of false positive rewards.

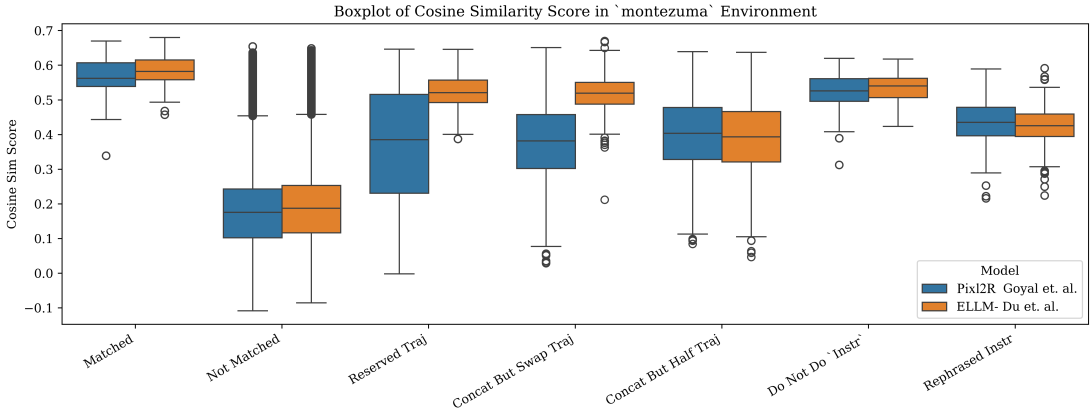


Figure 12: Cosine similarity scores for match, mismatch and manipulated trajectory-instruction pairs in Montezuma.

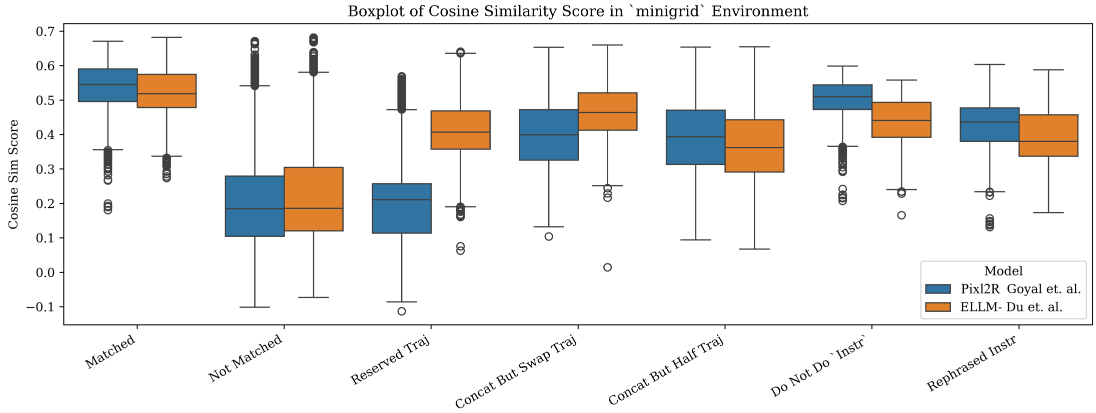


Figure 13: Cosine similarity scores for match, mismatch and manipulated trajectory-instruction pairs in Minigrid.

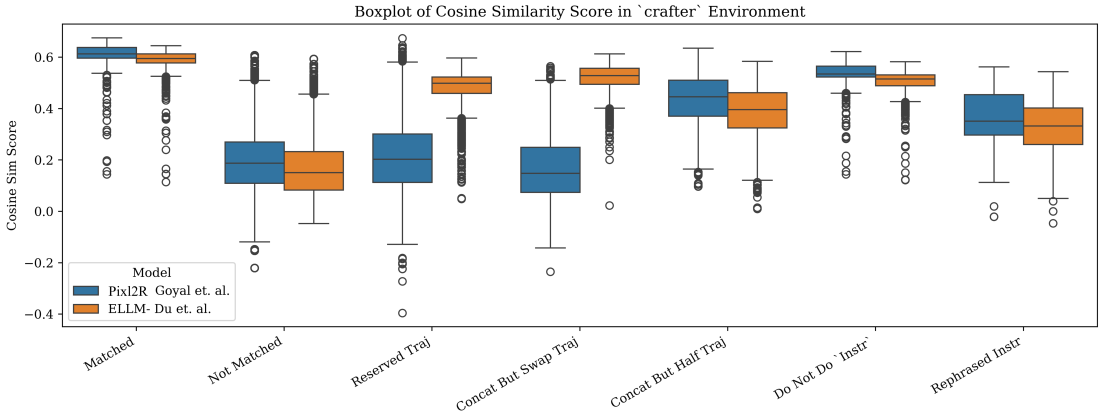


Figure 14: Cosine similarity scores for match, mismatch and manipulated trajectory-instruction pairs in Crafter.

A.7 Impact of Reward Model Quality on Policy Learning Efficiency: Supplementary Experiments and Quantitative Analysis in Montezuma

In our efforts to assess the impact of false positive rewards from auxiliary reward model without the interference of other factors such as domain shift, poor data quality, and errors from other issues such as the choice of multimodal architectures, we devised a **simulated** auxiliary reward model (also known as *oracle* auxiliary reward model) that access to internal state information from the game engine. The model compares the sequence of past actions and states of the agent with predefined target intermediate state sets that map to each instruction sentence. This is feasible in Montezuma's Revenge environment as we are able to access coordinate system information directly from the game engine. This access allows us to locate the current positional information of the agent and also label specific intermediate states as targets and assign rewards to the agent accordingly.

We therefore designed three types of simulated reward model:

- **Sim RM 1 (Perfect)** generates rewards accurately whenever the agent reaches the designated intermediate states. Furthermore, it strictly adheres to the chronological sequence of instructions; rewards for subsequent instructions are only awarded if all preceding instructions have been fulfilled.
- **Sim RM 2 (False Positive)** introduces a tolerance for false positive rewards but with reduced reward magnitudes, all while maintaining the temporal sequence. This is implemented by defining a radius σ, where if the agent enters a circle centered at the target state with a radius of σ, it receives a small amount of rewards.
- **Sim RM 3 (Temporal Insensitive False Positive)** disregards the chronological order of instructions, allowing rewards for later tasks even if earlier ones remain unfulfilled. However, note that fulfilling every sub-task will still result in the agent receiving the maximum total rewards. Therefore, in theory, the policy will eventually converge.

Results are reported in Table 7. Several important observations are as follows:

Table 7: Agent performance in Montezuma's Revenge, evaluated across three rooms with the goal state being the exit through a designated door. Metrics measured are the Area Under the Curve (AUC) for total reward (where higher is better) and success rate (SR) of reaching the goal state. The baseline is denoted as B, with $\star$ marking statistical significance ($p < 0.05$).

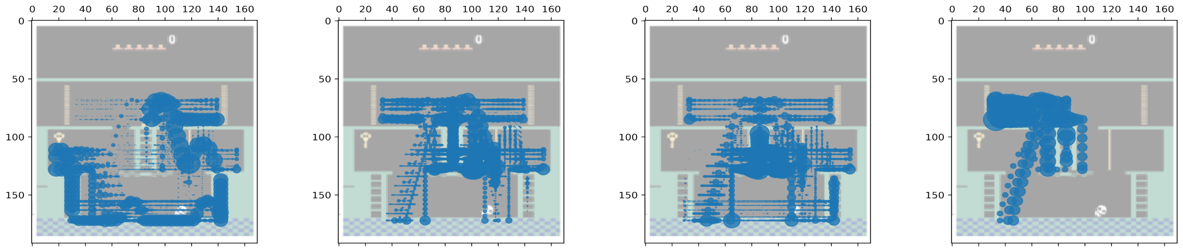
Model	AUC	SR
(1) PPO (Schulman et al., 2017)	all failed	0%
(2) PPO+RND (B) (Burda et al., 2018)	0.550±0.066	100%
(3) PPO + **Sim RM 1 (Perfect)**	0.287±0.048	100%
(4) PPO+RND + **Sim RM 1 (Perfect)**	0.608±0.073	100%
(5) PPO+RND + **Sim RM 2 (False Positive)**	0.183±0.187 $\star$	73.3%
(6) PPO+RND + **Sim RM 3 (Temp. Insen.)**	0.051±0.116 $\star$	16.7%

- Perfect auxiliary reward model have shown enhanced performance compared to weak RL baselines like PPO. While agents trained solely with PPO struggled to play Montezuma, incorporating Perfect auxiliary reward model into PPO (i.e., (3) in Table 7) did find the goal state, with the success rate increasing from 0% to 100%. However, we observed that the auxiliary reward model learns more slowly than the intrinsic reward model (see (2) in Table 7) when attempting to reach the goal state. This finding was initially surprising, but upon examining the agent movement heatmaps in Figure 15a and Figure 15b, an explanation becomes clear: the PPO+RND model discovers shorter paths to the goal state. The heatmap reveals that the PPO+RND agent learns to directly jump to a rope, bypassing the use of a ladder and conveyor belt as suggested by the expert instructions. This observation raises a concern about instruction-following based reward signals; they might limit the agent's exploration, confining it to states favored by the expert. If the quality of the expert's knowledge about the task is not optimal, this can result in the agent failing to compete with an exploration-based reward model.

- Again, we observed a remarkable synergy between the instruction-following-based reward model and the intrinsic reward model, as discussed in §7.1. As shown in (4) in Table 7, the PPO+RND + Sim RM 1 (Perfect) agent achieves the highest AUC score, demonstrating that it learns to reach the goal state faster than any other model.

- The involvement of false positive rewards significantly slows down the learning process of the model, as evidenced by entries (5) and (6) in Table 7. Additionally, both (5) and (6) show lower AUC scores compared to (2), which is the vanilla PPO+RND model. If we consider the vanilla PPO+RND model to represent a scenario with **full false negative** instruction-following rewards (since no instruction-following reward model is implemented here), then comparing the scores of (2) with (5) and (6) further indicates that false positive rewards are **more detrimental** than false negative ones.

- The RL + VLM reward approach completely failed to converge when the reward model disregarded temporal ordering, as evidenced by entry (6) in Table 7. The success rate had a significant drop to 16.7%, and even the RND intrinsic reward was unable to recover the success rate within the time budget. This illustrates the severe impact of false positive rewards due to temporal ordering insensitivity, an example of composition insensitivity discussed in §5; they trap the agent in a cycle of ineffective actions. However, note that if every sub-task is fulfilled, the agent will still receive the maximum total rewards, suggesting that, in theory, the policy could eventually converge. **Yet**, the practical implications of such false positives emphasize how they can drastically extend the learning period or lead to non-convergence in real-world scenarios where time or computational resources are limited.



(a) PPO+RND agent (same as + Sim RM (False Negative))

(b) PPO+RND + Sim RM 1 (Perfect)

(c) PPO+RND + Sim RM 2 (False Positive)

(d) PPO+RND + Sim RM 3 (Temporal Insensitivity)

Figure 15: Movement heatmap for PPO+RND agents when different simulated auxiliary instruction-following-based reward models are involved.

We also provided movement heatmap of different models for further qualitative analysis. As shown in Figure 15a, intrinsic reward model explored a shortcut towards the goal state by directly jump from the initial state to the rope, thereby learning to reach the goal state faster than other models.

Figures 15b and 15c do not exhibit a significant contrast, but they do reveal that the presence of false positive rewards leads the agent to more frequently visit dead ends, such as falling off cliffs. This behavior is consistent with the mechanics of the simulated false positive reward model, which defines a radius σ around target states. Within this radius, even if an agent falls off a cliff, it might receive rewards because its position could be close enough to the intended intermediate rewarding state. Consequently, agents had a chance of mistakenly perceiving falling off as advantageous.

Figure 15d illustrates the consequences when temporal ordering restrictions are removed: the learning agent persistently pursues rewards associated with the final instruction, "walk left to the door." However, simply executing this last instruction does not suffice for escaping the room because access to the door requires a key. This suggests that false positive rewards cause the agent to get trapped in local minima closer to s_0, indicating that deep RL can easily hack the reward system and effectively exploit shortcuts (Clark, 2016). A critical issue arises here: even with intrinsic rewards, the agent fails to escape these local optima, suggesting it has reached an equilibrium in the MDP where policy updates cease. This scenario is particularly relevant when the initial state distribution does not cover the entire state space. If the initial distribution does not have a positive probability for all states (i.e., $foralls \in \mathcal{S}, d_0(s) > 0$), it is not guaranteed that the RL algorithm will eventually reach the global optimum with maximum total rewards, as highlighted by (Agarwal et al., 2021).

A.8 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 (Sadinle et al., 2019):

Algorithm 2 Calculate Empirical Quantile ($\hat{q}$)

```
Require: Calibration set $\{\tau, l\}_n$, where $l$ is the instruction sentence, $\tau$ is the corresponding trajectory,
         and $n$ is the number of samples;
         Significance level $\alpha$;
         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.9 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 (Hafner, 2021). 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 × 512 steps × 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. (2021) 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.

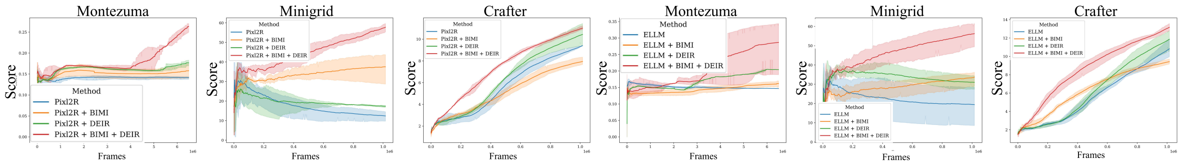


Figure 16: Besides the improvements of the score performance of agents across different environments with the BIMI reward function, it also collaborates well with intrinsic rewards. Combining both can lead to significant performance improvements

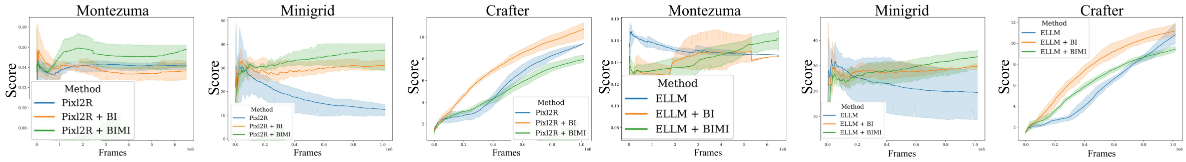


Figure 17: Ablation on the components of BIMI reward function. The binary reward (Bi) alone led to a 36.5% improvement compared to original models. Excluding Crafter, Mutual Information (MI) provided a 23% further improvement over Bi alone

A.10 Detailed Experiment Results of BIMI Reward Function

A.10.1 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-following agents in real-world, unfamiliar scenarios, where visual inputs often deviate from the VLMs' training distribution, leading to noisy reward signals.

A.10.2 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-following 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).

A.11 DEPRECATED Proof of the Reduction of Convergence Rate

WARNING: The following content is deprecated. This section contains the original convergence analysis for the ARR October Version, which has been identified as problematic by Reviewer nSDu. It is retained here solely for reference and backtracking purposes.

Formally, Abel et al. (2021) defined that:

Theorem A.7 (Abel et al. (2021)). *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.8. *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.

This proposition forms the basis of our theoretical analysis in Section A.11.

Below is the convergence analysis when the total return can be classified by returns of trajectories from acceptable policies $P(\tau \in \mathcal{T}_G \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G]$ versus returns of trajectories from unacceptable policies $P(\tau \in \mathcal{T}_B \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]$, where $\mathcal{T}_G \sim \Pi_G$, $\mathcal{T}_B \sim \Pi_B$ and $\Pi_B = \Pi \setminus \Pi_G$.

Note on Derivation: The proof presented here can be viewed as a reformulation of the Q-value function's gradient in the context of our specific problem setup. While this derivation may appear straightforward to readers well-versed in reinforcement learning, we include it to provide a clear mathematical foundation for our analysis of false positive rewards in the VLM-RL context. This formulation helps bridge the gap between standard RL theory and our specific problem domain on instruction-following RL with VLM-based rewards.

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

- **Critic**:

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi (\delta)^2 \tag{41}$$

  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**:

$$\theta \leftarrow \theta + \alpha_\theta \frac{Q_\phi(s,a) \nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \tag{42}$$

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

1. We use $\mathcal{T}_G$ to represent the set of acceptable trajectories and use $\mathcal{T}_B$ as a non-overlapping set of "not acceptable" trajectories, we can say:

$$\mathcal{T}_G \cap \mathcal{T}_B = \emptyset \tag{43}$$

   where $\mathcal{T}$ is the trajectory distribution space of the agent policy π_θ.

2. When we define $\mathcal{T}_B$ and $\mathcal{T}_G$ within the universe $\mathcal{T}$, we must account for the possibility of a remaining set to ensure a complete partition of $\mathcal{T}$. The remaining set $\mathcal{T} \setminus (\mathcal{T}_B \cup \mathcal{T}_G)\}$ can be interpreted as a trajectories that partially fulfill the instruction. Given our focus on the influence of false positive rewards within challenging sequential decision-making tasks, we assume that partially correct trajectories are also **unacceptable**. This stance is justified by the fact that partially correct trajectories can be viewed as false positives: they appear to be plausible behaviors but ultimately fail to fulfill the instructions. Rewarding such trajectories could reinforce incomplete or potentially dangerous behaviors. The errors discussed in Section 5 can be seen as instances of partially correct trajectories that are ultimately unacceptable. For example, incompletely following a safety-critical task such as "turn off the machinery before performing maintenance" could pose significant risks, and rewarding such behavior would be undesirable. Consequently, for the purposes of our analysis, we can assume:

$$\mathcal{T}_G \cup \mathcal{T}_B = \mathcal{T} \tag{44}$$

3. 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 $Q_{\phi_0}(s,a) = 0, \forall_{s \in S} \forall_{a \in A}$.

4. We assume that $|\mathcal{T}_G|$ and $|\mathcal{T}_B|$ is a fixed number predefined by the task environment while $\mathbb{E}_{\tau \in \pi_\theta}[G(\tau) \mid \tau \in \mathcal{T}_B]$ 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 $\|\mathbb{E}_{\tau \in \pi_\theta}[G(\tau) - Q_\phi(s,a)]\|^2$, the updated Q-value will approach as follows:

$$\begin{aligned}
&Q_\phi(s,a) \to \mathbb{E}[G(\tau)] && (45)\\
=&P(\tau \in \mathcal{T}_B \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B] + P(\tau \in \mathcal{T}_G \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G] \\
&+ P(\tau \in \mathcal{T} \setminus (\mathcal{T}_B \cup \mathcal{T}_G) \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T} \setminus (\mathcal{T}_B \cup \mathcal{T}_G)] && (46)\\
=&P(\tau \in \mathcal{T}_B \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B] + P(\tau \in \mathcal{T}_G \mid \theta) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G] \quad \text{[Assumption 2]} && (47)\\
& && (48)
\end{aligned}$$

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

$$\begin{aligned}
&\nabla_\theta Q_\phi(s,a)\pi_\theta(a \mid s) && (49)\\
=&\big(\nabla_\theta P(\tau \in \mathcal{T}_B \mid \theta_{\text{old}}) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]\pi_\theta(a \mid s)\big) \\
&+ \big(\nabla_\theta P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}}) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G]\pi_\theta(a \mid s)\big) && (50)\\
=&P(\tau \in \mathcal{T}_B \mid \theta_{\text{old}}) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]\nabla_\theta \pi_\theta(a \mid s) \\
&+ P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}}) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G]\nabla_\theta \pi_\theta(a \mid s) \quad \theta_{\text{old}} \text{ terms are constants w.r.t. } \theta && (51)\\
=&\big(1 - P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}})\big) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]\nabla_\theta \pi_\theta(a \mid s) \\
&+ P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}}) \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G]\nabla_\theta \pi_\theta(a \mid s) && (52)\\
=&\textit{const} \cdot \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]\nabla_\theta \pi_\theta(a \mid s) \\
&+ (\mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G] - \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]) \; P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}}) \; \nabla_\theta \pi_\theta(a \mid s) && (53)
\end{aligned}$$

Note the highlighted term in green ($P(\tau \in \mathcal{T}_G \mid \theta_{\text{old}})$) is the direction of the gradient ascent on parameter θ.

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. (2021)).

- In reference to the realizability condition implied by Assumption 3, the expressiveness of the policy class parameterized by θ is an underlying assumption for deep learning models, supported by the The Universal Approximation Theorem (Hornik et al., 1989).

Observation. Since the goal of the learning agent is to maximize $P(\tau \in \mathcal{T}_G \mid \theta)$ (i.e., to converge the agent policy to the distribution of acceptable trajectories), we can see that the second term provides the target direction with rate $(\mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_G] - \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B])$. Therefore, the ascent rate will decrease when the cumulative rewards from unacceptable trajectories (i.e., the false positive rewards) gets higher. In addition, the first term *const* $\cdot\, \mathbb{E}[G(\tau) \mid \tau \in \mathcal{T}_B]\nabla_\theta \pi_\theta(a \mid s)$ can be regarded as the deviation term of target direction. This formalization follows the intuition that the presence of false positive rewards can slow down the convergence rate of the learning agent.