VLM-GUIDED EXPERIENCE REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in Large Language Models (LLMs) and Vision-Language Models (VLMs) have enabled powerful semantic and multimodal reasoning capabilities, creating new opportunities to enhance sample efficiency, high-level planning, and interpretability in reinforcement learning (RL). While prior work has integrated LLMs and VLMs into various components of RL, the replay buffer, a core component for storing and reusing experiences, remains unexplored. We propose addressing this gap by leveraging VLMs to guide the prioritization of experiences in the replay buffer. Our key idea is to use a frozen, pre-trained VLM as an automated evaluator to identify and prioritize promising sub-trajectories from the agent's experiences. Across scenarios, including game-playing and robotics, spanning both discrete and continuous domains, agents trained with our proposed prioritization method achieve **15–57%** higher average success rates and improve sample efficiency by **35–55%** compared to previous approaches (Fig. 1).

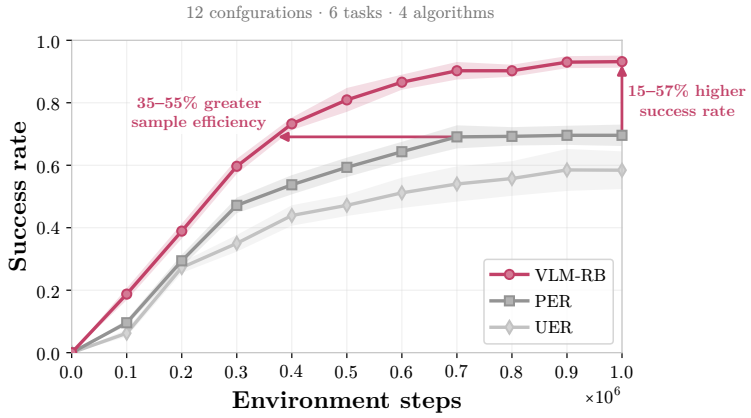


Figure 1: **VLM-RB learns more from the same interaction budget.** Mean success across 12 algorithm–task configurations, averaged across 5 seeds. Shading denotes standard error.

1 INTRODUCTION

Reinforcement learning (RL) has achieved impressive results across domains ranging from robotics to natural language systems (Levine et al., 2016; Ouyang et al., 2022) and logistics management (Tang et al., 2025). Despite this success, RL remains notoriously sample-inefficient, sensitive to hyperparameters, and heavily dependent on carefully designed reward functions (Kalashnikov et al., 2021; Kroemer et al., 2021; Zhu et al., 2020).

A well-established line of research—off-policy RL (Lin, 1992; Mnih et al., 2013)—addresses the efficiency bottleneck through experience reuse. During training, transitions are collected into a dataset known as a *replay buffer*, and sampled multiple times to update the policy. Ideally, a sampling mechanism would prioritize the most meaningful transitions: those that provide the richest learning signal, irrespective of when they were collected. The core challenge is to identify which experiences are truly meaningful for the learning process.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Conventional methods such as Prioritized Experience Replay (**PER**, Schaul et al. 2015b) approximate this ideal by prioritizing transitions with high temporal-difference (TD) error. The TD error measures the discrepancy between a bootstrapped value estimate and its target (Sutton et al., 1998). In effect, this approach promotes transitions which are misaligned with the agent's future outcome predictions. While TD error prioritization can correct value estimates, it is fundamentally limited by its lack of semantic awareness. It cannot distinguish transitions which reflect genuine progress toward task completion from those that do not.

This limitation is particularly evident in long-horizon, sparse-reward tasks, such as robotic manipulation in OGBench (Park et al., 2024). For instance, critical transitions—such as unlatching a door or grasping a tool—may yield low TD errors early in training. This is a consequence of delayed credit assignment preventing the reward signal from propagating to these earlier states. Conversely, visually distinct but task-irrelevant motions can yield large TD errors, despite contributing little to actual task completion.

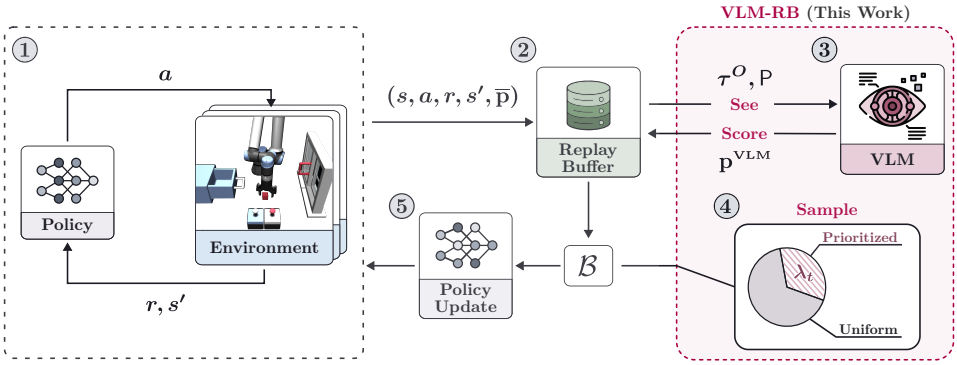


Figure 2: **System diagram:** (1) Data is collected with the current policy π_k interacting with multiple instances of the environment. (2) Transitions (s, a, r, s') are kept in a prioritized replay buffer with a provisional VLM score $\bar{\mathbf{p}}$. (3) Asynchronously, after each insertion, a VLM worker scores the corresponding rendered clip τ^O under a prompt P and writes the resulting score $\mathbf{p}^{\mathrm{VLM}}$ back to the replay buffer (see Section 3.1). (4) The learner samples a minibatch using the mixture distribution q_t which interpolates between uniform and VLM-prioritized replay (see Section 3.1). (5) The agent policy is updated to π_{k+1}.

To bridge this gap, we propose replacing heuristic inductive biases with external sources of semantic understanding. Recent literature highlights the potential of integrating large-scale Vision-Language Models (VLMs) with RL to leverage pre-trained world knowledge (Liang et al., 2024; Zhang et al., 2024). Because VLMs combine visual perception with language reasoning, they can interpret complex environments and mitigate RL-specific challenges—such as sparse rewards or inefficient exploration—by providing enriched context (Cao et al., 2024; Schoepp et al., 2025). This motivates our primary research question:

Can the semantic knowledge of pre-trained VLMs be used to prioritize meaningful experiences in replay buffers?

To answer this, we introduce **VLM-RB** (Fig.2), which integrates a pre-trained VLM directly into the experience prioritization pipeline. In contrast to heuristics relying on statistical proxies such as TD error, uncertainty (Carrasco-Davis et al., 2025), or density ratios (Zhao & Tresp, 2019), **VLM-RB** leverages the reasoning capabilities of VLMs to *score* experiences according to their semantic relevance.

We design this framework as a modular *plug-and-play* prioritization layer that can be integrated with any off-policy algorithm that uses a replay buffer. By using frozen off-the-shelf VLMs, we leverage strong semantic priors while avoiding the computational cost of fine-tuning or additional inference latency at deployment. Within the training pipeline, **VLM-RB** promotes efficiency by querying the VLM asynchronously, thereby sidestepping the blocking inference bottleneck that often constrains the throughput of LLM/VLM-enhanced RL methods.

We empirically show that pre-trained VLMs yield semantically grounded prioritization signals which are closely aligned with task objectives. Leveraging this semantic guidance, **VLM-RB** consistently outperforms existing baselines in both discrete game-playing and simulated continuous-control manipulation. Importantly, the moderate throughput overhead (about 12%) is more than offset by substantial gains in sample efficiency (35–55%) and average success rates (15–57%).

2 PRELIMINARIES

Off-Policy Reinforcement Learning. We consider the standard RL framework, where an agent interacts with an environment modeled as an infinite-horizon Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$, with state space $\mathcal{S}$, action space $\mathcal{A}$ (either discrete or continuous), transition kernel $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor $\gamma \in (0, 1)$ and initial-state distribution $\rho_0 \in \Delta(\mathcal{S})$. We further assume access to a rendering function $\psi : \mathcal{S} \to \mathcal{O} \subseteq \mathbb{R}^{H \times W \times 3}$ which maps states to $H \times W$-sized visual observations $o_t = \psi(s_t)$. The agent learns a stochastic policy $\pi_\theta : \mathcal{S} \to \Delta(\mathcal{A})$ to maximize the expected discounted return $J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ denotes a trajectory and $p_\pi(\tau)$ is the trajectory distribution induced by policy π.

To evaluate and improve policies, value functions are used to estimate expected returns. The *action-value function* (Q-function) $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represents the expected discounted return from state s after taking action a and then following policy π: $Q^\pi(s, a) = \mathbb{E}_{\tau \sim p_\pi(\tau | s_0 = s, a_0 = a)} \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. In modern RL methods, Q^π is typically a learned function (represented by a neural network), denoted the *critic*.

In off-policy RL, rather than exclusively using on-policy data, optimization is performed using previously collected experience stored in a *replay buffer* (Mnih et al., 2013). Let $\mathcal{D}_t$ denote the replay buffer at time t. We sample training tuples $(s, a, r, s') \in \mathcal{D}_t$ according to a time-dependent distribution $q_t \in \Delta(\mathcal{D}_t)$. In standard *uniform* replay (**UER**), $q_t(i) = 1/|\mathcal{D}_t|$ for all $i \in \mathcal{D}_t$. This formulation highlights that the optimization dynamics depend on the evolving sampling distribution q_t.

Experience Replay Prioritization. As opposed to uniform replay, prioritized replay methods bias retrieval toward transitions expected to yield a stronger learning signal. For clarity, we formalize this process as three distinct steps: *scoring*, *prioritization*, and *sampling*. *Scoring* assigns each transition $i \in \mathcal{D}_t$ a scalar value $\mathbf{p}_i \in \mathbb{R}$ reflecting its heuristic utility. *Prioritization* maps these raw scores into a probability distribution over the buffer, $q_t^{\mathrm{P}} \in \Delta(\mathcal{D}_t)$. *Sampling* then draws a minibatch of indices $\{i_k\}_{k=1}^{B}$ from a target distribution q_t, or a mixture of several such distributions.

Prioritized Experience Replay (**PER**, Schaul et al. 2015b) is a specific instance of this framework. Its scoring function utilizes the temporal-difference (TD) error magnitude, $\mathbf{p}_i = |\delta_i| + \varepsilon$ (where $\varepsilon > 0$ ensures non-zero probability), where δ_i is the discrepancy between the value estimate and its target, namely $\delta_i = Q_\theta(s_t, a_t) - (r + \gamma Q_\theta(s_{t+1}, a'))$, where θ are the critic parameters and a' is the action predicted by the current policy in s_{t+1}. In Q-learning methods, a' is the argmax action, and in actor-critic methods for continuous control, it is $\pi(s_{t+1})$. The prioritized distribution is defined as $q_t^{\mathrm{P}}(i) \propto \mathbf{p}_i^\alpha$, where $\alpha \in [0, 1]$ determines the degree of prioritization. While **PER** effectively focuses updates on transitions with larger TD errors, it relies on value estimates which may be arbitrary and noisy early in training, particularly in sparse-reward settings.

Vision–Language Models. Recent advances in large-scale representation learning have given rise to *Vision–Language Models* (VLMs), which learn aligned embeddings of visual and textual modalities through joint pre-training on large-scale image–text corpora (Radford et al., 2021; Jia et al., 2021; Zhai et al., 2023; Li et al., 2023). These models have demonstrated strong generalization across a wide range of downstream tasks, including zero-shot recognition, goal specification, and reward inference (Ahn et al., 2022; Rocamonde et al., 2023). While standard VLMs process static images, *Video Question Answering* (Video-QA) models extend these capabilities to ingest a temporal sequence of frames (a video clip) along with a natural-language query, aggregating visual information across time to reason about events, motion, and causal relationships. Concretely, for the rest of this paper, we employ Perception-LM (Cho et al., 2025), a state-of-the-art open-source Video-QA family of models with multiple model sizes (1B/3B/8B).

3 VLM-RB

In this section, we present **VLM-RB** (Fig.2 and Algorithm 1), a plug-and-play framework designed for any off-policy algorithm that leverages a replay buffer. The central idea is to use a VLM to assign semantic scores to sub-trajectories, subsequently biasing the sampling distribution toward higher-scoring experiences during policy optimization.

3.1 Scoring, Prioritization, and Sampling

We now define how semantic scores are extracted from collected data, how these scores induce a prioritization distribution, and how **VLM-RB** samples from this distribution.

Scoring. **VLM-RB** uses a pre-trained frozen VLM to extract semantic scores from collected experiences. To enable this, we first construct temporal visual sequences from the agent's history. Formally, let $\tau_i^O = (o_i, o_{i+1}, \ldots, o_{i+L-1})$ denote a visual clip comprising L rendered frames. The VLM maps this clip and a text prompt P to a scalar score:

$$\mathbf{p}_i^{\text{VLM}} = f_{\text{VLM}}(\tau_i^O, \mathsf{P}) \in \mathbb{R}, \tag{1}$$

where different choices of f_{VLM} are detailed in Appendix E. The prompt P directs the VLM's scoring mechanism, leveraging the model's inherent world knowledge to identify meaningful behaviors even in the absence of dense reward signals. While this interface supports the injection of detailed and task-specific priors, we find that the VLM's intrinsic scene understanding is sufficient for our purposes. We therefore employ a general goal-agnostic, domain-generic prompt (see Appendix C for details).

This effectively labels useful sequences without the need for hand-crafted task-specific definitions. Crucially, because the VLM is frozen and its evaluation depends only on static visual content, each clip is scored exactly once. This offers a significant efficiency advantage over methods such as **PER**, which require ongoing priority updates as the value function evolves.

A natural question is whether scoring individual frames, rather than clips, would suffice. We argue that single frames are fundamentally limited by *semantic ambiguity*. Consider Fig. 3: a single observation of a robotic gripper hovering above an object is visually identical across two vastly different temporal contexts, the onset of a successful grasp or the aftermath of a failed attempt. Without temporal context a VLM cannot disambiguate these scenarios, potentially assigning high scores to frames which actually belong to failure modes. By scoring sub-trajectories instead, we increase the likelihood of the VLM having sufficient temporal information to distinguish meaningful behaviors from failures. While a sliding-window variant could potentially provide denser labels, we leave this (computationally intensive) alternative for future work.

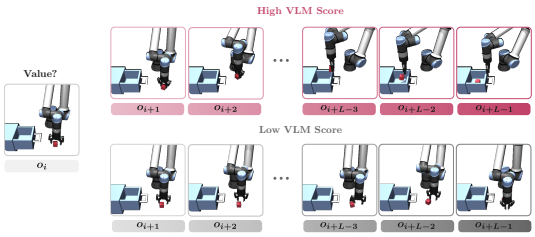


Figure 3: **Temporal context resolves visual ambiguity.** From the initial state o_i (left), multiple futures are possible. The top sub-trajectory shows a successful grasp, while the bottom shows stagnation. By scoring sub-trajectories rather than single frames, the VLM has sufficient context to distinguish meaningful progress from failure modes.

Prioritization. We propagate the VLM score of each clip to all transitions within that clip and combine it multiplicatively with the latest TD error. This defines our priority signal, which is converted into a distribution:

$$\mathbf{p}_i = \mathbf{p}_i^{\text{VLM}} |\delta_i|, \qquad q_t^{\text{P}}(i) = \frac{(\mathbf{p}_i + \varepsilon)^\alpha}{\sum_{j \in \mathcal{D}_t} (\mathbf{p}_j + \varepsilon)^\alpha}. \tag{2}$$

The two factors carry complementary information: $\mathbf{p}_i^{\text{VLM}}$ marks which experiences are semantically consistent with task success, whereas $|\delta_i|$ marks where the value function is still inaccurate.

Sampling. If we were to sample only from q_t^{P}, updates would concentrate on the highest-scoring transitions. Because the VLM is an external prior and may score useful transitions inaccurately, we instead preserve full-buffer coverage by mixing VLM-guided prioritization with uniform replay:

$$q_t(i) = \lambda_t q_t^{\mathrm{P}}(i) + \frac{1-\lambda_t}{|\mathcal{D}_t|}, \tag{3}$$

where $\lambda_t \in [0,1]$ controls the strength of the VLM guidance. In practice, each batch draws a λ_t fraction from q_t^{P} and the remainder uniformly. We use a linear warm-up schedule: starting with $\lambda_0 = 0$ (pure uniform sampling), we gradually anneal to $\lambda_{\max} = 0.5$ over the first half of training. The warm-up favors broad coverage early in training, while the uniform component limits the influence of inaccurate VLM scores throughout training. Removing this component is detrimental in our ablation (Section 4.4).

3.2 Efficient Implementation

Incorporating a VLM into the RL loop introduces significant computational overhead relative to standard components. We address this with two key design choices. First, we leverage the decoupling between data collection and policy optimization in off-policy learning to asynchronously score experiences. As illustrated in Fig. 2, the VLM interacts with the replay buffer in the background, ensuring that policy optimization is never blocked by inference latency. Until a clip's VLM score becomes available, $\mathbf{p}_i^{\mathrm{VLM}}$ is replaced by the running mean of previously returned VLM scores. Second, we use a lightweight 1B parameter model (Cho et al., 2025) (see Appendix F.5).

4 Experiments

In this section, we evaluate the efficacy of **VLM-RB** in leveraging semantic priors for efficient exploration. We first ask: does the VLM signal provide meaningful guidance for exploration? To answer this, we analyze its correlation with learned value estimates and its dependence on visual semantics (Section 4.2). Next, we benchmark **VLM-RB** against **UER**, **PER**, and alternative prioritization schemes, evaluating both performance and sample efficiency across a range of discrete and continuous control tasks (Section 4.3). Finally, we examine which design choices most affect the performance of **VLM-RB**, focusing on the prioritization signal, the sampling mixture, and VLM model size (Section 4.4).

4.1 Experimental Setup

Tasks. We evaluate **VLM-RB** on two domains with discrete and continuous action spaces: (i) `DoorKey` from `MiniGrid` (Chevalier-Boisvert et al., 2023), using grid sizes `8x8`, `12x12`, and `16x16` to vary exploration difficulty; and (ii) `scene` from OGBench (Park et al., 2024), using the predefined tasks 3, 4, and 5, which require increasingly long-horizon compositional manipulation (unlocking/locking and coordinated object placement). In all experiments, agents receive *state-based* observations (rather than pixels) to minimize perception-related confounders and isolate the effect of replay prioritization on exploration and sample efficiency. More details about the various tasks are provided in Appendix B.

Baselines. We compare **VLM-RB** to two standard replay sampling methods: Uniform Experience Replay (**UER**), which samples transitions uniformly from the replay buffer, and **PER**, which prioritizes sampling by TD-error. We report **UER** and **PER** comparisons for four different algorithms: DQN (Van Hasselt et al., 2016), IQN (Dabney et al., 2018), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018). In addition, to compare **VLM-RB** against alternative replay prioritization methods, we run a focused ablation for DQN comparing to Attentive Experience Replay (AER, Sun et al. 2020), Experience Replay Optimization (ERO, Zha et al. 2019), and Reducible Loss Prioritization (ReLO, Sujit et al. 2023). Implementation details for all baselines are provided in Appendix E.

Metrics. We report the Success Rate (SR), defined as the fraction of episodes in which the agent completes the task within the maximum episode length, $T_{\max}$. The Average Success Rate (ASR) is the mean SR over $N = 32$ evaluation episodes for a given seed. Final results are the ASR averaged over $M = 5$ seeds, with the standard error of the mean (SEM) shown as shaded regions in plots.

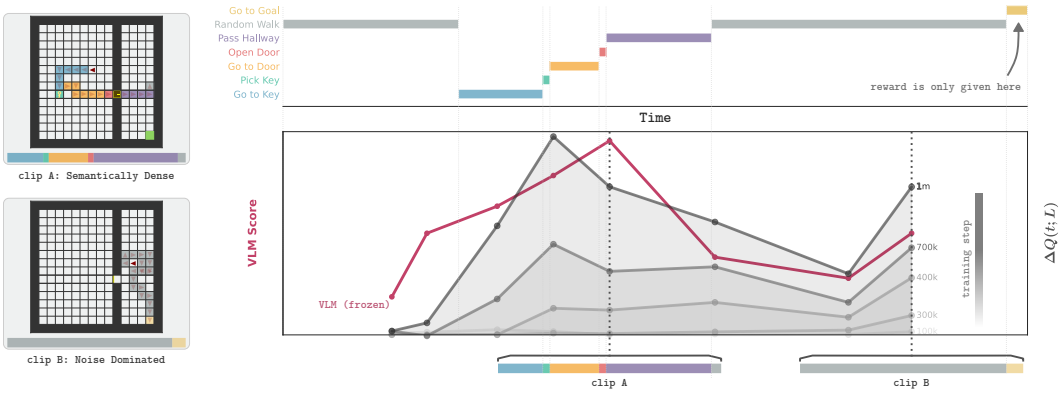


Figure 4: **Frozen VLM scoring qualitatively anticipates learned value.** We visualize a single reference episode in `MiniGrid/DoorKey-16x16` which contains both goal-directed actions and random noise. **Left:** Visualizations of two 32-frame sub-trajectories: **`clip A`** captures a semantically relevant sequence, while **`clip B`** shows a mostly random walk which eventually reaches the goal. **Top right:** A timeline of ground-truth events; sparse reward is only received at the final goal. Both right panels share the reference-episode timestep on the x-axis. **Bottom right:** The rose curve shows the frozen VLM score for $L = 32$-frame clips. The gray curves show the temporal value difference, $\Delta Q(t; L) = Q(t + \frac{L}{2}) - Q(t - \frac{L}{2})$, calculated from critic checkpoints at increasing training steps (light to dark).

4.2 Do VLMs Encode Useful Semantic Priors?

Before turning to downstream RL results, we first ask whether the frozen VLM offers a signal that is actually useful for prioritizing data. To illustrate the semantic grounding of our prioritization, we compare the frozen VLM scores with critic value differences (ΔQ) as learning progresses. Concretely, we track how the value estimates of the critic evolve on a fixed reference episode, using checkpoints from a successful training run. As shown in Fig. 4, the critic's estimates are initially flat and uninformative (light gray lines). Over the course of training, the critic gradually learns to assign high value to the same semantic events (such as picking up a key or opening a door) that the VLM identified from the outset. This observation suggests that **VLM-RB** can immediately identify semantically relevant data, enabling faster learning than approaches which must wait for the critic to converge.

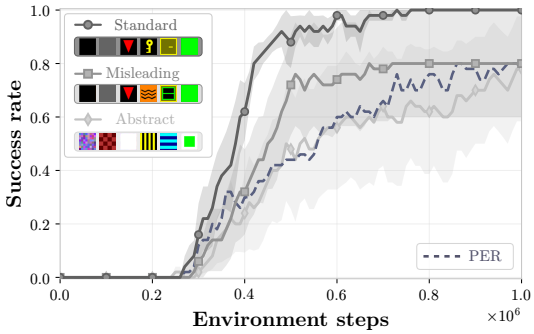


Figure 5: **Success depends on alignment with semantic priors.** We compare the performance of **VLM-RB** across frames rendered with *Standard* visuals (dark circles), *Misleading* swapped sprites (medium squares), and *Abstract* textures (light diamonds). The dashed line indicates the ASR of **PER**. Crucially, the agent's actual observations and the underlying MDP are identical across all settings; only the visual input to the VLM is altered (see Appendix F.11 for visual samples of these modifications). The significant performance drop in the *Misleading* and *Abstract* settings confirms that our method relies on the VLM correctly recognizing specific semantic objects (e.g., keys and doors) to accelerate learning.

Having observed qualitative alignment between the VLM and a well-trained Q-value predictor on this reference episode, we next ask whether the signal reflects interpretable semantic priors.

To test whether **VLM-RB** specifically leverages the VLM's pre-trained visual semantics, we employ the "Modified Game" paradigm (Dubey et al., 2018). We modify only the rendered frames used for VLM scoring, leaving the underlying MDP and agent observations untouched. We consider three settings: (i) the *Standard* game, with unaltered visuals; (ii) the *Misleading* game, where key objects are swapped (for example, traps appear as goals); and (iii) the *Abstract* game, where objects are replaced with random noise patterns (See Appendix F.11 for visual examples).

As shown in Fig. 5, the *Misleading* and *Abstract* variants significantly slow learning and increase variance. Notably, in the *Abstract* setting, performance degrades to the level of **PER** (dashed line). Because the underlying MDP remains unchanged, this degradation demonstrates that the VLM's prioritization is only effective when the visual input aligns with natural semantic priors.

In summary, these results confirm that **VLM-RB** scores are indicative of meaningful semantic behavior. We now turn to the question: does this semantic guidance actually yield improved sample efficiency and asymptotic performance in off-policy RL?

Table 1: Improvement with respect to baselines. **Performance** denotes the mean per-seed best ASR ($M = 5$ seeds). **Sample efficiency** tracks the steps required to reach the *baseline's* best performance; unreached targets are assigned the 1M-step horizon. Values in parentheses indicate the relative improvement over the baseline. Both metrics are averaged across the aggregated algorithms. The corresponding training curves are provided in Fig. 11.

		vs. **UER**		*vs.* **PER**	
Algorithm	**Level**	**Performance** (↑)	**Sample Efficiency** (↓)	**Performance** (↑)	**Sample Efficiency** (↓)
DQN+IQN (`DoorKey`)	8×8	0% (1.00/1.00)	22% (139K/178K)	0% (1.00/1.00)	26% (139K/187K)
	12×12	61% (1.00/0.62)	57% (277K/640K)	22% (1.00/0.82)	37% (292K/467K)
	16×16	283% (0.92/0.24)	53% (420K/893K)	92% (0.92/0.48)	51% (305K/622K)
SAC+TD3 (`Scene`)	3	0% (1.00/1.00)	40% (213K/354K)	0% (1.00/1.00)	20% (213K/266K)
	4	22% (1.00/0.82)	62% (295K/776K)	2% (1.00/0.98)	38% (394K/634K)
	5	147% (0.89/0.36)	55% (366K/819K)	68% (0.89/0.53)	37% (508K/805K)

4.3 Main Results: VLMs are Useful for RL Data Prioritization

We evaluate **VLM-RB** and baselines across both discrete and continuous environments, as detailed in Section 4.1. Table 1 summarizes the aggregate improvements in both performance and sample efficiency. **VLM-RB** matches or exceeds every baseline in success and improves sample efficiency in 23 of 24 comparisons. The largest gains appear in the most challenging settings: on `DoorKey-16x16`, **VLM-RB** achieves a **+283.3%** improvement over **UER** and **+91.7%** over **PER**; on `Scene-5`, the improvement over **UER** is **+147.2%** and **+67.9%** over **PER**. Even in cases where baselines eventually solve the task, **VLM-RB** achieves substantially better sample efficiency, reducing the number of steps needed to reach baseline performance by as much as **62.0%**.

Comparison to Baselines. Figure 6 expands this comparison, showing that **VLM-RB** is the only method to consistently solve the task when compared to alternative prioritization schemes (AER, ERO, ReLo). The baselines struggle primarily due to the combination of sparse rewards and long sequential dependencies. ERO and ReLo fail to prioritize task-relevant transitions because their feedback signals are either dominated by noise or remain zero throughout the long pre-success phase, while similarity-based methods like AER over-sample current states and neglect critical past dependencies (such as key acquisition), impeding temporal credit assignment over long horizons. Further analysis is provided in Appendix D.

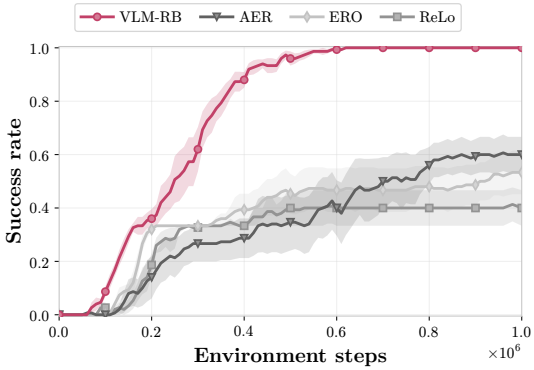


Figure 6: **VLM-RB overcomes the failure modes of heuristic prioritization in sparse-reward tasks.** Aggregated success rates on `MiniGrid/DoorKey` across grid sizes. While alternative methods (AER, ERO, ReLo) fail as they depend on dense rewards or local similarity metrics, **VLM-RB** successfully bridges the long-horizon dependencies. Curves show the mean across 5 seeds, with shaded standard errors.

Training Time. Finally, we address the computational trade-off. Although VLM inference introduces a moderate throughput overhead (approximately 12% wall-clock time per step, see Appendix F.12), this cost is outweighed by the reduction in training steps. In the vast majority of configurations (23/24), **VLM-RB**

reduces the absolute wall-clock time to reach peak performance by up to **67%**, with speedups exceeding **20–40%** in the more challenging exploration settings; the sole exception is a 23% slowdown (full results in Table 16, Appendix G).

4.4 Which Design Choices Matter? Mixing and Scale

Sampling Mixture. A central component of our approach is to mix VLM-scored samples with uniformly sampled data, allowing the external semantic prior to guide replay while preserving uniform coverage. To assess its importance, we vary the maximal mixing coefficient, $\lambda_{\max}$, holding all other parameters fixed. The no-mixture configuration fails to solve the task, whereas the scheduled configurations succeed, showing the importance of mixing VLM-prioritized and uniform replay during training; Appendix F.3 provides the full analysis, and Appendix F.4 reports the realized sampling frequencies under the default $\lambda_{\max} = 0.5$ mixture. Appendix F.8 directly tests this safeguard under controlled VLM label corruption.

VLM Size. We also ablated the VLM size and found that scaling the model beyond 1B parameters dramatically increases computational cost without yielding consistent improvements in downstream RL performance (see Appendix F.5 for the trade-offs between throughput, memory, and performance).

5 Related Work

Prioritized Experience Replay. Extensive research has focused on defining *which* transitions maximize learning utility. The seminal work, **PER** (Schaul et al., 2015b), prioritizes transitions with high TD error, serving as a proxy for "surprising" events. Subsequent methods have proposed alternative utility metrics based on state similarity (Sun et al., 2020), potential loss reduction (Sujit et al., 2023), target quality/discrepancy criteria (Kumar et al., 2020), or by explicitly learning a replay policy (Zha et al., 2019). However, these approaches typically derive priorities from internal, training-dependent signals (e.g., value estimates or TD errors), which can be noisy or undefined early in training. In contrast, **VLM-RB** leverages a pre-trained VLM to assign semantic priorities, yielding a robust utility score from the very first update. Orthogonal to the choice of *what* to prioritize, several works modify *how* transitions are sampled. Hong et al. (2022) argue for enforcing the ordering of updates within a collected episode, to better align with the Bellman loss that is common in off-policy learning. Another example, Lahire et al. (2021), focuses on the "staleness" of prioritization scores and suggests a double-sampling approach. These directions are largely complementary to **VLM-RB** and can be combined with VLM-based priorities. Furthermore, because **VLM-RB** computes priorities from a frozen VLM, it naturally avoids the score staleness issue addressed by the latter. A distinct line of work instead alters the *optimization objective* itself, deriving reweighting schemes from f-divergence regularization (Li et al., 2024), trust-region constraints (Novati & Koumoutsakos, 2019), or density-ratio estimation (Liu et al., 2021; Sinha et al., 2022). **VLM-RB** changes no loss function, and so remains modular and compatible with a wide range of existing algorithms.

LLM/VLM-Enhanced RL. Recent work has increasingly used foundation models to improve RL by injecting semantic knowledge into the learning loop. Broadly, these methods leverage LLMs/VLMs to (i) define richer learning signals (reward supervision), (ii) propose plans and exploration targets (high-level guidance), or (iii) assist action selection (policy components). Comprehensive surveys can be found in Cao et al. (2024) and Schoepp et al. (2025). A major thread treats LLMs and VLMs as sources of reward. Off-the-shelf VLMs have been used as zero-shot reward scorers to judge state–goal alignment (Rocamonde et al., 2023); LLMs can synthesize dense shaping functions from natural-language task descriptions (Xie et al., 2023); and several methods replace or augment human feedback by using VLM/LLM judgments (absolute ratings or pairwise preferences) to train reward models (Fu et al., 2024; Luu et al., 2025; Singh et al., 2025; Ghosh et al., 2025). Taken together, these approaches reframe reward design as semantic labeling, allowing pre-trained models to provide informative supervision in settings where environment rewards are sparse or absent. A second strand uses foundation models for planning and exploration. LLMs decompose instructions into temporally-extended plans grounded in affordances (Ahn et al., 2022; Huang et al., 2022), and guide exploration by proposing semantically diverse subgoals or novelty-oriented objectives using learned embeddings (Ma et al., 2025; Gupta et al., 2022). This high-level guidance is particularly useful in settings where unguided exploration rarely visits meaningful states. A third direction integrates LLMs/VLMs more directly into control, either as action advisors or as

policy components. Examples range from generating executable policy code (Liang et al., 2022) to large-scale vision–language–action (VLA) models that map multimodal inputs to actions (Zitkovich et al., 2023). Despite this breadth, most prior work uses these models to shape what the agent optimizes (rewards), where it searches (plans and exploration), or how it acts (policy), which commonly introduces additional training stages, fine-tuning, or computationally expensive inference. In contrast, **VLM-RB** targets the replay buffer—the core mechanism enabling sample reuse in off-policy RL—a component which remains underexplored for semantic foundation-model supervision. Moreover, as our experiments show, open-source pre-trained VLMs can already yield consistent gains when used to prioritize replay, without any additional model training or fine-tuning.

6 Conclusions

In this work, we presented **VLM-RB**, a framework that integrates the semantic reasoning capabilities of pre-trained VLMs directly into the experience replay mechanism. By shifting prioritization from statistical proxies such as TD-error to semantic evaluations of task progress, **VLM-RB** effectively identifies and promotes meaningful experiences even in sparse-reward regimes where traditional metrics fail. Our empirical results demonstrate that this approach significantly improves both sample efficiency and asymptotic performance across discrete and continuous domains, without requiring fine-tuning or gradient updates to the VLM.

Limitations. While our approach yields substantial improvements, it is important to clarify the specific assumptions and constraints under which these gains are realized. First, **VLM-RB** fundamentally relies on the assumption that task progress can be reliably inferred from visual observations. As a result, **VLM-RB** does not apply to domains where the underlying state lacks a visual representation, such as non-spatial biological systems or abstract network control tasks. In these settings, the VLM cannot ground its reasoning. Second, querying the VLM introduces additional computational overhead. Empirically, we found that this overhead reduced throughput by approximately 12% in our experiments. It is important to note that this computational cost is not intrinsic to the method; the precise overhead depends on hardware configuration and implementation choices, so the trade-off may vary across different setups[1]. Finally, while our approach uses a goal-agnostic, domain-generic prompt, performance is ultimately bound by the VLM's zero-shot visual reasoning capabilities, which may degrade in visually unstructured environments or require more careful prompt engineering and score calibration.

Future Work. Our findings suggest several concrete directions for future research. A natural extension is the application of **VLM-RB** to Goal-Conditioned RL (Schaul et al., 2015a), where the textual description of the current goal can be injected into the VLM prompt to dynamically score alignment, or hindsight textual goals could promote meaningful sub-goals which may have some semantic relation but no direct relevance to the current task (Luu & Yoo, 2021; Jiang et al., 2019). Another promising direction is to use the replay buffer as a curriculum mechanism, by evolving the prompt over the course of training, prioritizing specific skills such as "open drawer" in the early stages and later shifting focus to more compositional behaviors. Finally, the prioritization scheme could be extended beyond single VLM scoring, for example, by incorporating more robust inference techniques such as majority voting or LLM-as-a-Judge.

AI use statement

Generative AI tools were used to assist with planning and reviewing ablation experiments, editing analysis and plotting code, organizing and reformatting experimental artifacts, checking numerical consistency, and revising the manuscript for clarity and formatting. Experimental observations were produced by the reported runs, not generated by AI. The authors inspected and tested AI-assisted code, verified reported numbers and claims against raw logs and exported artifacts, and reviewed all AI-assisted text and figures. The authors take responsibility for the final content of this work.

[1] Our implementation did not employ aggressive optimization techniques (e.g., quantization or dedicated serving stacks), suggesting that this overhead could be further minimized.

Ethics statement

This work advances sample-efficient reinforcement learning in domains where visual priors and semantic reasoning offer a tangible advantage by leveraging vision-language models to prioritize experiences. Like other improvements to RL training pipelines, the technique we introduce is method-level and not tied to a specific deployment; it could, in principle, accelerate beneficial applications (e.g., robotics for assistive tasks) or, like any RL advance, downstream applications with negative externalities. We do not foresee specific risks beyond those generally associated with RL research.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch. *arXiv preprint arXiv:2306.00577*, 2023.

Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Yue Chen, Guolong Liu, Gaoqi Liang, Junhua Zhao, Jinyue Yan, and Yun Li. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Rodrigo Carrasco-Davis, Sebastian Lee, Claudia Clopath, and Will Dabney. Uncertainty prioritized experience replay. *arXiv preprint arXiv:2506.09270*, 2025.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394, 2023.

Jang Hyun Cho, Andrea Madotto, Effrosyni Mavroudi, Triantafyllos Afouras, Tushar Nagarajan, Muhammad Maaz, Yale Song, Tengyu Ma, Shuming Hu, Hanoona Rasheed, Peize Sun, Po-Yao Huang, Daniel Bolya, Suyog Jain, Miguel Martin, Huiyu Wang, Nikhila Ravi, Shashank Jain, Temmy Stark, Shane Moon, Babak Damavandi, Vivian Lee, Andrew Westbury, Salman Khan, Philipp Krähenbühl, Piotr Dollár, Lorenzo Torresani, Kristen Grauman, and Christoph Feichtenhofer. Perceptionlm: Open-access data and models for detailed visual understanding. *arXiv:2504.13180*, 2025.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L Griffiths, and Alexei A Efros. Investigating human priors for playing video games. *arXiv preprint arXiv:1802.10217*, 2018.

Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. Furl: Visual-language models as fuzzy rewards for reinforcement learning. *arXiv preprint arXiv:2406.00645*, 2024.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Udita Ghosh, Dripta S Raychaudhuri, Jiachen Li, Konstantinos Karydis, and Amit Roy-Chowdhury. Preference vlm: Leveraging vlms for scalable preference-based reinforcement learning. *arXiv preprint arXiv:2502.01616*, 2025.

Tarun Gupta, Peter Karkus, Tong Che, Danfei Xu, and Marco Pavone. Foundation models for semantic novelty in reinforcement learning. *arXiv preprint arXiv:2211.04878*, 2022.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Zhang-Wei Hong, Tao Chen, Yen-Chen Lin, Joni Pajarinen, and Pulkit Agrawal. Topological experience replay. *arXiv preprint arXiv:2203.15845*, 2022.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International conference on machine learning*, pp. 4904–4916. PMLR, 2021.

Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Dmitry Kalashnikov, Jake Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Scaling up multi-task robotic reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021.

Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research*, 22(30):1–82, 2021.

Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in neural information processing systems*, 33:18560–18572, 2020.

Thibault Lahire, Matthieu Geist, and Emmanuel Rachelson. Large batch experience replay. *arXiv preprint arXiv:2110.01528*, 2021.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Changling Li, Zhang-Wei Hong, Pulkit Agrawal, Divyansh Garg, and Joni Pajarinen. Roer: Regularized optimal experience replay. *arXiv preprint arXiv:2407.03995*, 2024.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PMLR, 2023.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Zijing Liang, Yanjie Xu, Yifan Hong, Penghui Shang, Qi Wang, Qiang Fu, and Ke Liu. A survey of multimodel large language models. In *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering*, pp. 405–409, 2024.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

Xu-Hui Liu, Zhenghai Xue, Jingcheng Pang, Shengyi Jiang, Feng Xu, and Yang Yu. Regret minimization experience replay in off-policy reinforcement learning. *Advances in neural information processing systems*, 34:17604–17615, 2021.

Tung M Luu and Chang D Yoo. Hindsight goal ranking on replay buffer for sparse reward environment. *IEEE Access*, 9:51996–52007, 2021.

Tung Minh Luu, Younghwan Lee, Donghoon Lee, Sunho Kim, Min Jun Kim, and Chang D Yoo. Enhancing rating-based reinforcement learning to effectively leverage feedback from large vision-language models. *arXiv preprint arXiv:2506.12822*, 2025.

Runyu Ma, Jelle Luijkx, Zlatan Ajanović, and Jens Kober. Explorllm: Guiding exploration in reinforcement learning with large language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9011–9017. IEEE, 2025.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Guido Novati and Petros Koumoutsakos. Remember and forget for experience replay. In *International Conference on Machine Learning*, pp. 4851–4860. PMLR, 2019.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. *arXiv preprint arXiv:2310.12921*, 2023.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Sheila Schoepp, Masoud Jafaripour, Yingyue Cao, Tianpei Yang, Fatemeh Abdollahi, Shadan Golestan, Zahin Sufiyan, Osmar R Zaiane, and Matthew E Taylor. The evolving landscape of llm-and vlm-integrated reinforcement learning. *arXiv preprint arXiv:2502.15214*, 2025.

Anukriti Singh, Amisha Bhaskar, Peihong Yu, Souradip Chakraborty, Ruthwik Dasyam, Amrit Bedi, and Pratap Tokekar. Varp: Reinforcement learning from vision-language model feedback with agent regularized preferences. *arXiv preprint arXiv:2503.13817*, 2025.

Samarth Sinha, Jiaming Song, Animesh Garg, and Stefano Ermon. Experience replay with likelihood-free importance weights. In *Learning for Dynamics and Control Conference*, pp. 110–123. PMLR, 2022.

Shivakanth Sujit, Somjit Nath, Pedro Braga, and Samira Ebrahimi Kahou. Prioritizing samples in reinforcement learning with reducible loss. *Advances in Neural Information Processing Systems*, 36:23237–23258, 2023.

Peiquan Sun, Wengang Zhou, and Houqiang Li. Attentive experience replay. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5900–5907, 2020.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28694–28698, 2025.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. Experience replay optimization. *arXiv preprint arXiv:1906.08387*, 2019.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5625–5644, 2024.

Rui Zhao and Volker Tresp. Curiosity-driven experience prioritization via density estimation. *arXiv preprint arXiv:1902.08039*, 2019.

Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

A VLM-RB

Algorithm 1 VLM-prioritized replay buffer

Require: prompt P; clip length L.
replay buffer $\mathcal{D}$; clip buffer $\mathcal{C}$; queues $\mathcal{Q}_{\text{in}}$ and $\mathcal{Q}_{\text{out}}$.

function VLMWORKER
  while true **do**
    $(\texttt{idxs}, \tau^O) \leftarrow \textbf{pop}(\mathcal{Q}_{\text{in}})$
    $\mathbf{p}^{\text{VLM}} = f_{\text{VLM}}(\tau^O, \mathsf{P}) \in \mathbb{R}$ (Equation 1)
    push$(\mathcal{Q}_{\text{out}}, (\texttt{idxs}, \mathbf{p}^{\text{VLM}}))$

Init: launch VLMWORKER asynchronously.
while TRAINING **do**
  ▷ (1) Env step
  Step env to obtain (s, a, r, s') and visual observation $o = \psi(s)$
  $\texttt{idx} \leftarrow \text{INSERT}(\mathcal{D}, (s, a, r, s'; \mathbf{p}^{\text{VLM}} \leftarrow \bar{\mathbf{p}}))$
  $\mathcal{C} \leftarrow \mathcal{C} \cup (\texttt{idx}, o)$
  ▷ (2) Enqueue clips (streaming)
  if $|\mathcal{C}| = L$ **or** `terminated` **or** `truncated` **then**
    $\texttt{idxs}, \tau^O \leftarrow \{(\texttt{idx}_i, o_i)\}_{i=0}^{|\mathcal{C}|-1}$
    push$(\mathcal{Q}_{\text{in}}, (\texttt{idxs}, \tau^O))$
    $\mathcal{C} \leftarrow \emptyset$
  ▷ (3) Apply VLM scores (drain output queue)
  while $\mathcal{Q}_{\text{out}}$ not empty **do**
    $(\texttt{idxs}, \mathbf{p}^{\text{VLM}}) \leftarrow \textbf{pop}(\mathcal{Q}_{\text{out}})$
    for all $\texttt{idx} \in \texttt{idxs}$ **do**
      $\text{SETVLMSCORE}(\mathcal{D}, \texttt{idx}, \mathbf{p}^{\text{VLM}})$
    $\bar{\mathbf{p}} \leftarrow \text{CMA}(\bar{\mathbf{p}}, \mathbf{p}^{\text{VLM}})$
  ▷ (4) Sample and learn
  $\mathcal{B} \sim q_t$ (Equation 3)
  UPDATELEARNER$(\mathcal{B})$; update λ_t

In Algorithm 1, $\mathcal{C}$ denotes a temporary clip buffer, $\mathcal{Q}_{\text{in}}$ and $\mathcal{Q}_{\text{out}}$ are asynchronous communication queues, and CMA refers to the cumulative moving average of returned VLM scores.

B ENVIRONMENT DETAILS

In this section, we describe the environments used in our experiments.

MiniGrid / DoorKey. We use the `DoorKey` tasks from the MiniGrid suite (Chevalier-Boisvert et al., 2023) with sizes `8x8`, `12x12`, and `16x16`. Each episode requires the agent to complete a sequence of subtasks: (i) navigate to and pick up the key, (ii) reach the locked door and open it using the `toggle` action (which is only possible while holding the key), and (iii) proceed to the goal tile. This structure enforces temporal dependencies and compositional reasoning. The reward function is sparse: the agent receives zero reward at all intermediate steps and a positive reward only upon reaching the goal, with the reward magnitude linearly decaying with the number of steps taken, following the standard MiniGrid protocol. Episodes terminate either upon successful completion or when the maximum step limit is reached.

State observations are provided in a symbolic format, where each $s_t \in \{0, 1, \ldots\}^{N \times N \times 3}$ encodes the full grid. The three channels correspond to (i) object indices, (ii) color indices, and (iii) a state channel that includes both the door state and the agent's orientation. The action space is discrete with $|\mathcal{A}| = 5$, consisting of turn left, turn right, move forward, pick up, and toggle actions.

OGBench / Scene. We use the `scene-play` manipulation environment from OGBench (Park et al., 2024), which is a MuJoCo-based tabletop task involving a UR5e arm with a parallel gripper. The agent interacts with a set of objects: a cube, a sliding drawer, a sliding window, and two toggle buttons. Each button controls the lock state of either the drawer or the window, introducing dependencies that require the agent to execute explicit unlock, manipulate, and (re-)lock sequences to achieve many goals. Episodes are capped at a horizon of 750 steps. To facilitate learning, we employ potential-based

reward shaping with a per-step living cost, defined as:

$$r_t = -1 \; + \; \gamma\,\phi(s_{t+1}) - \phi(s_t), \qquad \phi(s) = \frac{1}{K}\sum_{i=1}^{K} \mathbb{I}\left[\text{subgoal}_i(s)\right].$$

Here, $\phi(s)$ denotes the fraction of satisfied subgoals, where each subgoal is a task-specific predicate over cube placement, button states, and the positions of the drawer and window. An episode terminates either when all subgoals are satisfied ($\phi(s_t) = 1$) or when the maximum horizon is reached.

Observations are provided as full state vectors, $s_t \in \mathbb{R}^{40}$, and actions are specified as continuous end-effector controls, $a_t \in \mathbb{R}^5$.

To address the exploration challenges inherent in these long-horizon tasks, we initialize the replay buffer with a lightweight warm-start of 10 demonstration episodes, which are held fixed across all methods. This small set of demonstrations provides minimal task-relevant coverage, while leaving the remainder of the training protocol unchanged. Warm-starting the buffer this way follows common practice in demonstration-guided RL (Hester et al., 2018; Vecerik et al., 2017; Nair et al., 2018). All methods share the same 10 episodes, task reward, and shaping; only the replay priority differs. `MiniGrid/DoorKey` uses no demonstrations.

We evaluate on a set of predefined Scene goals (tasks 3–5), which are designed to progressively increase the degree of temporal composition required for successful completion.

Task 3 (rearrange-medium). The agent must move the cube to a specified tabletop location, open the drawer, close the window, and terminate with both the drawer and window unlocked. Notably, the window begins in a locked and open state, so the policy must first unlock it before closing, while simultaneously coordinating the manipulation of the drawer and the relocation of the cube.

Task 4 (put-in-drawer). The agent is required to place the cube inside the drawer and terminate with the drawer closed and unlocked, while ensuring the window remains locked. This sequence involves unlocking the drawer, opening it, inserting the cube, and then closing the drawer.

Task 5 (rearrange-hard). The agent must place the cube inside the (closed) drawer and leave the window open, while ensuring that both the drawer and window are locked at the end of the episode. Achieving this goal requires the agent to execute both unlock and relock sequences, coordinate drawer opening and closing, and correctly position the cube.

C PROMPTS

Task prompts used in our experiments. We use a binary "success visible" query with a strict output format. Both prompts are *goal-agnostic*: neither names the goal, the objects involved, or any success predicate, so the same text is reused unchanged across all `DoorKey` grid sizes (8×8, 12×12, 16×16) and across `Scene` tasks 3–5, whose goals differ. They are also *domain-generic* in the sense that the only domain-dependent content is the list of visual cues in the `Scene` variant, which describes manipulation in general rather than any particular task. We report the first phrasing we wrote in each case; neither was tuned against downstream RL performance, and no prompt selection was performed.

1. `MiniGrid/DoorKey`: *"Does this clip contain a clear instance of goal satisfaction anywhere in it? If no visible success occurs, answer No. Do not guess. Output exactly Answer: Yes or Answer: No."*
2. `OGBench/Scene`: *"Is there at least one clear instance of goal satisfaction in these frames? Look for contact + displacement consistent with the goal (lift off surface, place into receptacle, open/close articulation, move to target zone). Do not guess. If not visible, answer No. Output exactly Answer: Yes or Answer: No."*

Meta-prompt for generating task prompts. To facilitate applying our procedure to new environments, we use the following meta-prompt to generate a task prompt from a task identifier and optional human context. The meta-prompt encourages generic, visually grounded success criteria and enforces a strict output schema.

Meta-prompt template for generating task prompts

User: You are an expert in Reinforcement Learning and Visual Language Models. I will provide a short clip from an agent rollout. Your job is to write a single text prompt that asks a VLM to output a **binary** judgment about whether **goal satisfaction / competent task progress is clearly visible** in the clip.

Requirements:

- Keep the prompt **environment-agnostic**: rely on visible physics and outcomes rather than simulator-specific rules.
- Specify what to look for in broad categories when helpful (e.g., contact + displacement for manipulation; reaching a target region for navigation).
- Explicitly instruct: **do not guess**; if success is not clearly visible, answer No.
- End the prompt with the string: **Output exactly Answer: Yes or Answer: No.**

Examples (few-shot):

- **Task:** Minigrid-doorkey
  Prompt: "Does this clip contain a clear instance of goal satisfaction anywhere in it? If no visible success occurs, answer No. Do not guess. Output exactly Answer: Yes or Answer: No."
- **Task:** OGBench-scene-task-4
  Prompt: "Is there at least one clear instance of goal satisfaction in these frames? Look for contact + displacement consistent with the goal (lift off surface, place into receptacle, open/close articulation, move to target zone). Do not guess. If not visible, answer No. Output exactly Answer: Yes or Answer: No."

Current request:
Task Name: `{{TASK_ID}}`
Optional Human Context: `{{HUMAN_CONTEXT}}`

Example: prompt families for a new benchmark. As an illustration, we can apply the meta-prompt to a suite such as DeepMind Control by grouping tasks by their visible success patterns (locomotion / balance / interaction) and instantiating one generic prompt per group.

Example: prompt families for DeepMind Control Suite

User: Group DeepMind Control tasks by shared *visible* success patterns and propose one generic success-detection prompt per group (not task-specific). Each prompt must end with: "Output exactly Answer: Yes or Answer: No."

Model: We used Gemini 3 Pro to generate these prompts. Here is the partition into three semantic groups based on the **nature of the movement**, with a generalized "vibe-check" prompt for each.

Group 1: Going Somewhere (Locomotion)
The Semantic Goal: The agent must make progress across the environment. The VLM needs to distinguish between an agent that is struggling/falling (chaos) and one that is moving purposefully (control).

The Group: `cheetah-run`, `dog-walk`, `dog-trot`, `dog-run`, `fish-swim`, `hopper-hop`, `humanoid-walk`, `humanoid-run`, `quadruped-walk`, `quadruped-run`, `walker-walk`, `walker-run`.

The Prompt:

> "Does the agent demonstrate controlled, purposeful movement through the environment? Look for the agent making progress in a specific direction (locomotion) rather than just flailing in place, falling over, or crashing. The movement should look stable and competent. If the agent falls, flips, or fails to move effectively, answer No. Output exactly Answer: Yes or Answer: No."

Group 2: Defying Gravity (Stabilization)
The Semantic Goal: The agent is fighting gravity. Success is defined by **stillness** or **verticality**. Whether it is a pole swinging up or a dog standing still, the "vibe" of success is reaching a stable, upright state and holding it.

The Group: `acrobot-swingup`, `cartpole-balance`, `cartpole-balance_sparse`, `cartpole-swingup`, `cartpole-swingup_sparse`, `dog-stand`, `hopper-stand`, `humanoid-stand`, `walker-stand`, `pendulum-swingup`.

The Prompt:

> "Does the agent successfully achieve and maintain a stable, upright posture against gravity? Look for the agent balancing itself or an object vertically. Success means the agent (or the object it controls) is standing up or swinging up to a balanced position and staying there without collapsing, spinning out of control, or falling over. If it looks unstable or falls, answer No. Output exactly Answer: Yes or Answer: No."

Group 3: Touching the Thing (Interaction)
The Semantic Goal: This is about precise manipulation. The agent is not trying to go somewhere or just stand up; it is trying to force an interaction between its end-effector and a target object.

The Group: `ball_in_cup-catch`, `finger-spin`, `finger-turn_easy`, `finger-turn_hard`, `reacher-easy`, `reacher-hard`.

The Prompt:

> "Does the agent successfully interact with the target object in this scene? Look for the agent reaching a target, catching an object, or spinning a mechanism as intended. There should be clear, controlled contact or manipulation consistent with solving a task. If the agent misses the target, drops the object, or flails without making meaningful contact, answer No. Output exactly Answer: Yes or Answer: No."

D BASELINES

`DoorKey` presents a particularly challenging regime, characterized by extremely sparse rewards (identically zero until termination) and a long-horizon, sequential dependency structure (key to door to goal). In such settings, replay schemes relying on dense feedback, informative TD statistics, or a well-defined similarity metric are fundamentally misaligned with the task: they fail to prioritize transitions relevant to solving the task, particularly throughout the extended pre-success phase.

Experience Replay Optimization (ERO) (Zha et al., 2019). ERO optimizes a learned rejection sampler using a REINFORCE-style update, where the scalar reward is defined as the change in evaluation return. In `DoorKey`, evaluation returns remain identically zero for a substantial portion of training (reflecting the absence of successes), which in turn implies $r_{\text{replay}} \approx 0$ and results in near-zero gradients for the replay policy over extended periods. Furthermore, the replay-policy features used here, $(r_i, |\delta_i|, t_i/T_{\max})$, are weakly informative under sparse rewards: $r_i = 0$ for almost all transitions and $|\delta_i|$ is largely determined by bootstrap noise in the early stages of learning. As a result, the rejection policy effectively acts as an untrained stochastic filter for much of the training. When successes eventually occur, r_{replay} becomes highly variable, leading to unstable and non-stationary updates which can bias acceptance toward incidental correlates, such as late timesteps. The top-up rule, which fills with the highest-P_i candidates when acceptance is low, further concentrates replay on a narrow subset of transitions. This reduces diversity and fails to provide a consistent signal for solving the task.

ReLo (Reducible Loss Prioritization) (Sujit et al., 2023). ReLo prioritizes transitions based on the difference between the magnitudes of online and target TD residuals, $\mathbf{p}_{\text{ReLo}} = \max(0, |\delta_\theta| - |\delta_{\theta^-}|) + \varepsilon$. In the sparse-reward `DoorKey` setting, both residuals are typically dominated by bootstrapping error rather than meaningful reward propagation throughout the prolonged pre-success phase. Moreover, the use of hard target updates periodically synchronizes the online and target networks, further diminishing any systematic separation between $|\delta_\theta|$ and $|\delta_{\theta^-}|$. Consequently, $|\delta_\theta| - |\delta_{\theta^-}|$ is often small or negative and is clipped to approximately ε, so the sampling distribution effectively reverts toward uniform, yet still incurs the variance and bias tradeoffs associated with prioritized replay.

Attentive Experience Replay (AER) (Sun et al., 2020). AER selects samples closest (in a frozen embedding space) to the agent's current state, thereby inducing a strongly local replay distribution. In `DoorKey` this notion of locality is fundamentally misaligned with the task structure. Once trajectories reach later phases (near the door or goal), nearest-neighbor replay disproportionately samples those regions and neglects earlier structural dependencies, such as key acquisition, thereby impeding temporal credit assignment across the long horizon. This effect is further amplified by the use of a randomly initialized frozen encoder: the resulting metric primarily captures superficial spatial proximity in the grid encoding, rather than functional or task-relevant similarity. As a result, the selection rule acts as a myopic location-based filter, rather than an attentive semantic sampler.

E Implementation Details

To ensure a fair comparison, all baselines are implemented with the same backbone architecture, optimizer, and training schedule as **VLM-RB**. Unless otherwise specified, each method employs a **PER**-style replay buffer parameterized by exponents α and β.

E.1 Baseline Implementations

Uniform Experience Replay. In this setting, each transition i in the buffer is sampled with equal probability, $q_t^{\mathrm{U}}(i) = 1/N_{\text{curr}}$, where N_{curr} denotes the current buffer size.

PER (Prioritized Experience Replay). Here, the priority assigned to each transition i is given by $\mathbf{p}_i = |\delta_i| + \varepsilon$, where δ_i denotes the most recent TD-error for that transition, and $\varepsilon = 10^{-6}$ guarantees that every transition can be sampled. The probability of sampling transition i is then defined as

$$q_t^{\mathrm{P}}(i) = \frac{\mathbf{p}_i^{\alpha}}{\sum_k \mathbf{p}_k^{\alpha}},$$

with α determining the extent to which prioritization influences sampling. To account for the bias from prioritized sampling, importance-sampling (IS) weights are computed as

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{q_t^{\mathrm{P}}(i)} \right)^{\beta}.$$

These weights are normalized by $1/\max_j(w_j)$ prior to being used in the loss.

Experience Replay Optimization (ERO). ERO (Zha et al., 2019) replaces standard replay sampling with a learned rejection policy applied to uniformly drawn candidates. The replay policy is parameterized as an MLP ϕ_ψ with two hidden layers of 64 units each (ReLU activations, sigmoid output), mapping transition features to a retention probability.

$$\mathbf{p}_i = \phi_\psi(r_i,\ |\delta_i|,\ t_i/T_{\max}) \in (0, 1),$$

where r_i is the immediate reward, $|\delta_i|$ is the absolute TD-error stored in the buffer (refreshed after each gradient step), and $t_i/T_{\max}$ is the normalized within-episode timestep. For each new transition, the absolute TD-error is initialized to $|\delta| = 1.0$ upon insertion. At each update, a candidate pool of size $N_{\text{cand}} = 4B$ is drawn uniformly from the buffer, where B is the batch size. Each candidate is independently retained with probability $\mathbf{p}_i$. If fewer than B transitions are accepted, the batch is completed by selecting the remaining candidates with the highest $\mathbf{p}_i$, ensuring a batch of size B. As in Zha et al. (2019), importance sampling is not applied. The replay policy is updated using a REINFORCE-style gradient, where evaluation returns serve as the reward signal. Let $\bar{R}$ represent a moving average of the evaluation return. After each evaluation, the reward for the replay policy is computed as $r_{\text{replay}} = \bar{R}_{\text{current}} - \bar{R}_{\text{previous}}$, and $\bar{R}$ is updated accordingly. With $\mathcal{B}$ denoting the selected batch, the replay policy loss is

$$\mathcal{L}_{\text{ERO}} = -\, r_{\text{replay}} \sum_{i \in \mathcal{B}} \log \mathbf{p}_i,$$

where gradients are applied only to ψ.

ReLo (Reducible Loss Prioritization). ReLo (Sujit et al., 2023) assigns priority to each transition based on a reducible-loss score, which is computed from the online and target Q-networks. Given a transition (s, a, r, s'), the target value is defined as $y = r + \gamma \max_{a'} Q_{\theta^-}(s', a')$. The online and target TD-errors are then $\delta_{\text{online}}(s, a) = Q_\theta(s, a) - y$ and $\delta_{\text{target}}(s, a) = Q_{\theta^-}(s, a) - y$, respectively. The sampling priority is

$$\mathbf{p}_{\text{ReLo}}(s, a) = \max(0,\ |\delta_{\text{online}}(s, a)| - |\delta_{\text{target}}(s, a)|) + \varepsilon, \qquad \varepsilon = 10^{-6}.$$

Each new transition is initialized with the maximum priority value among all transitions currently in the buffer. The method sets $\alpha = 0.6$ and initializes $\beta = 0.4$, annealing β linearly to 1.0. Standard importance-sampling weights are applied.

Attentive Experience Replay (AER). AER (Sun et al., 2020) selects transitions using an attentive sampling mechanism. A dedicated encoder ϕ is constructed with the same backbone architecture as the Q-network. This encoder is randomly initialized and remains fixed throughout training. At

each training step t, a candidate pool of size $N_{\text{cand}} = \lfloor \lambda_t B \rfloor$ is sampled uniformly from the buffer, where λ_t decays linearly from $\lambda_0 = 4$ to 1 over the course of training. If the candidate pool size N_{cand} is less than or equal to the batch size B, the method reverts to the default sampling strategy. Given the current state s_{curr} and a set of candidate states $\{s_i\}$, distances are computed in the frozen embedding space as $d(s_{\text{curr}}, s_i) = \|\phi(s_{\text{curr}}) - \phi(s_i)\|_2^2$. The B candidates with the smallest distances are then selected deterministically to form the training batch. Following Sun et al. (2020), importance sampling is disabled.

E.2 Agent Architectures

MiniGrid agents (DQN / IQN). Both DQN and IQN are trained on symbolic Minigrid observations $s_t \in \{0, \dots\}^{N \times N \times 3}$, which encode object identities, colors, and state information such as door status and agent orientation. To process these inputs, we employ a shared encoder which first embeds each channel, then applies a residual dilated CNN with three blocks. The resulting features are aggregated using both average and max global pooling, yielding a 256-dimensional representation. This design aims to capture both local and global spatial structure in the environment. For DQN, the 256-dimensional feature vector is passed through a small MLP, which outputs scalar Q-values for each discrete action. In IQN, we replace the scalar output head with an implicit quantile head. Specifically, we embed sampled quantile fractions using cosine functions, project these embeddings to match the feature dimension, and combine them element-wise with the state features. The resulting representations are mapped to per-action quantile values. We optimize using the quantile Huber loss and select actions using either Double-DQN-style (Van Hasselt et al., 2016) or target-network-based strategies, as described in Appendix E. This approach allows IQN to model the full distribution over returns, rather than just the mean.

OGBench Scene agents (SAC / TD3). For `OGBench/Scene`, both SAC and TD3 operate on flattened state-based observations $s_t \in \mathbb{R}^{40}$. We adopt a unified architecture across tasks: the actor is a fully-connected network with three hidden layers of width 512 and ReLU activations. This network outputs either a tanh-squashed Gaussian policy (for SAC) or a deterministic tanh policy (for TD3), enabling flexible action selection in continuous spaces. The critic is implemented as an ensemble of $N = 10$ Q-functions, each parameterized by a 3-layer MLP that receives the concatenated state and action as input and outputs a scalar value. For target computation, we randomly select $M = 2$ ensemble members and aggregate their predictions using a mean for SAC and a minimum for TD3. This ensemble approach is intended to improve stability and reduce overestimation bias. For actor updates, we use the same Q-function ensemble, aggregating Q-values according to the actor reduction rule. In all reported experiments, SAC uses the mean and TD3 the minimum over ensemble members. This ensures consistency between actor and critic updates.

E.3 Replay and Scoring Implementation

Replay buffer and prioritization. All methods are implemented with a two-branch replay ensemble: one prioritized branch and one uniform branch which share an underlying storage, implemented using `ReplayBufferEnsemble` (Bou et al., 2023) with mixture weights controlled by the sampling ratio schedule in Tables 2–3. For the **PER** baseline, we assign priorities using the standard PER formula $\mathbf{p}_i = |\delta_i| + \varepsilon$, where δ_i is the TD error. When importance-sampling corrections are enabled, we apply them as usual. New transitions are initially assigned a small default priority, and we update their TD-errors after each gradient step to ensure accurate prioritization. In **VLM-RB**, we augment each transition with both a TD-based metric and a VLM-based score. Their product defines the priority signal $\mathbf{p}_i$, and the standard **PER** transformation produces q_t^{P} (Equation 2); Appendix F.1 ablates the two factors. A transition enters the buffer with the provisional VLM score $\bar{\mathbf{p}}$, the cumulative moving average (CMA) of the VLM scores returned so far, so that a transition awaiting its score uses the current average. Once the worker returns the score of the clip containing i, $\mathbf{p}_i^{\text{VLM}}$ overwrites $\bar{\mathbf{p}}$. As in **PER**, $|\delta_i|$ is refreshed at every replay of i, and the product is formed at sampling time.

VLM worker. Decoupling scoring from the learner also makes **VLM-RB** agnostic to the policy input modality — the VLM requires rendered frames, while the policy may operate on arbitrary observation spaces such as low-dimensional states — and lets a single VLM instance serve multiple parallel environments. The VLM worker uses Perception-LM-1B (Cho et al., 2025) with a fixed prompt to detect binary success. It processes non-overlapping clips of $L = 32$ frames. The clip buffer

is flushed at termination or truncation, so clips never cross episode boundaries; a shorter final clip is left-padded. We fix $L = 32$ across all experiments and do not tune it. We define the raw score r^{VLM} as the first generated token's "Yes"/"No" mass ratio, summing the probability assigned to all "Yes" token variants (m_{Yes}) and all "No" variants (m_{No}). The binarized instantiation used in the main experiments is

$$r^{\mathrm{VLM}} = \frac{m_{\mathrm{Yes}}}{m_{\mathrm{Yes}} + m_{\mathrm{No}}} \in [0, 1],$$
$$\mathbf{p}^{\mathrm{VLM}} = f_{\mathrm{VLM}}(\tau^O, \mathsf{P}) = \mathbf{1}[r^{\mathrm{VLM}} > \tau], \qquad \tau = 0.5. \tag{4}$$

The ratio itself requires no calibration data, but a hard threshold depends on the score scale of the chosen VLM and prompt. We use $\tau = 0.5$ for the default Perception-LM prompt; transferring the hard-threshold variant to another VLM or prompt may require a small labeled calibration corpus, while using r^{VLM} directly avoids threshold calibration. Appendix F.2 evaluates both choices. The resulting scores are asynchronously streamed back to update the replay buffer, ensuring the main learner is not blocked during this process.

Bounded influence of the VLM score. Because a fraction $(1-\lambda_t)$ of every batch is drawn uniformly, the VLM changes how often a transition is replayed, never whether it can be. For any f_{VLM}, transition i is sampled at

$$A_t(i) = \frac{q_t(i)}{1/N_t} = (1 - \lambda_t) + \lambda_t N_t \, q_t^{\mathrm{P}}(i) \tag{5}$$

times its uniform rate, where N_t is the number of transitions in the buffer. The first term is a floor: with $\lambda_{\max} = 0.5$, every transition retains at least half its uniform probability regardless of its score. A binary score makes the role of the numerical floor explicit. Transitions with zero product signal receive prioritized mass ε^α / Z_t, where Z_t is the denominator in Equation 2, and are therefore amplified slightly above $(1 - \lambda_t)$. When this floor is negligible relative to the positive product signals, the $|P_t|$ promoted transitions absorb nearly all of the prioritized branch and their average amplification approaches $(1 - \lambda_t) + \lambda_t N_t / |P_t|$, while all other transitions approach $(1 - \lambda_t)$. This is roughly half the factor purely prioritized sampling would apply, and the warm-up keeps it near 1 early in training, when the buffer is small and errors would be most costly.

E.4 HYPERPARAMETERS

Table 2: Hyperparameters and implementation details for OGBench experiments. Shared parameters are listed once; divergent parameters are compared side-by-side.

Hyperparameter	**TD3**	**SAC**
Network Architecture		
Hidden Dimensions	[512, 512, 512]	
Q-Network Layer Norm	True	True
Optimization		
Critic Learning Rate	3×10^{-4}	
Actor Learning Rate	3×10^{-4}	1×10^{-3}
Alpha Learning Rate	–	3×10^{-4}
Batch Size	256	
Discount Factor (γ)	0.99	
Target Smoothing (τ)	0.005	
Max Grad Norm	10.0	
Learning Starts	10,000 steps	
Ensemble & Update Ratios		
Ensemble Size (N)	10	
Subsampled Q-Networks (M)	2	
Critic UTD Ratio	4	
Actor UTD Ratio	2	
Target Q Reduction	min	mean
Actor Q Reduction	min	mean
Data & Replay Buffer		
Replay Buffer Size	1×10^6	
Expert Demos	10	
Algorithm Specifics		
Policy Update Frequency	2	1
Exploration Noise (std)	0.1	–
Target Policy Noise (std)	0.2	–
Noise Clip	0.5	–
Entropy Coeff. (α)	–	1.0 (Initial)
Target Entropy Scale	–	0.5
Auto Entropy Tuning	–	True
PER *Baseline Settings*		
PER Alpha (α)	0.7	
PER Beta (β)	1.0	
Importance Sampling	False	
VLM-RB *Settings*		
VLM Model	Facebook Perception-LM-1B	
Priority Signal	$\mathbf{p}^{\text{VLM}} \cdot \lvert\delta\rvert$	
Priority Mode	Hard Threshold (> 0.5)	
Trajectory Length (L)	32 Frames	
Importance Sampling	False	
Prioritized Sampling Ratio (λ_0 and $\lambda_{\max}$)	Annealed $0.0 \rightarrow 0.5$	
Annealing Schedule (T_{schedule})	Linear over 500k steps	

Table 3: Hyperparameters and implementation details for MiniGrid experiments. Shared parameters are listed once; divergent parameters are compared side-by-side.

Hyperparameter	**DQN**	**IQN**
Optimization		
Learning Rate	4×10^{-5}	
Batch Size	128	
Discount Factor (γ)	0.95	
Target Update Frequency	1,000 steps	
Target Update Rate (τ)	1.0 (Hard Update)	
Learning Starts	500 steps	
Train Frequency	4 steps	
Max Grad Norm	1.0	
Exploration (Epsilon-Greedy)		
$\text{eps}_{\text{start}}$	1.0	
eps_{end}	0.05	
Exploration Fraction	0.5 (First 50% of training)	
Algorithm Specifics		
Double DQN	Enabled	
Noisy Nets	–	Disabled
Num Quantiles (Policy)	–	32
Num Quantiles (Train/Target)	–	64
Num Cosine Basis Functions	–	64
Huber Kappa (κ)	–	1.0
PER *Baseline Settings*		
PER Alpha (α)	0.7	
PER Beta (β)	1.0	
Importance Sampling	True	
Prioritized Sampling Ratio	1.0 (Always Prioritized)	
VLM-RB *Settings*		
VLM Model	Facebook Perception-LM-1B	
Priority Signal	$\mathbf{p}^{\text{VLM}} \cdot \lvert\delta\rvert$	
Priority Mode	Hard Threshold (> 0.5)	
Trajectory Length (L)	32 Frames	
Importance Sampling	False	
Prioritized Sampling Ratio (λ_0 and $\lambda_{\max}$)	Annealed $0.0 \rightarrow 0.5$	
Annealing Schedule (T_{schedule})	Linear over 500k steps	

F Ablations

F.1 Prioritization Variants

To compare the two factors in Equation 2, we evaluate uniform replay, TD-only priority, VLM-only priority, and their product on `DoorKey-16x16`/DQN and `Scene-5`/SAC. All VLM-bearing arms use the binarized instantiation with $\tau = 0.5$. Each cell contains 5 seeds.

Table 4: Prioritization variants on `DoorKey-16x16`/DQN and `Scene-5`/SAC (5 seeds per cell). Entries report ASR at 1M steps, followed in parentheses by the steps required to reach the best ASR attained by **UER** and **PER**, respectively. "Never" means that the target was not reached within 1M steps.

Priority	DoorKey-16x16	Scene-5
UER	0.20 (920K, never)	0.44 (900K, never)
PER (TD-only)	0.76 (330K, 850K)	0.62 (570K, 970K)
VLM-only	1.00 (310K, 410K)	0.88 (470K, 670K)
$\mathbf{p}^{\mathrm{VLM}}\lvert\delta\rvert$	1.00 (310K, 410K)	0.94 (320K, 440K)

The VLM-only priority exceeds both baselines on both tasks. Multiplying it by $|\delta|$ leaves the DoorKey result unchanged and improves Scene-5 from 0.88 to 0.94 ASR, while reducing the steps to **PER**'s best performance from 670K to 440K. We therefore use the product in the main experiments.

F.2 Threshold Selection and the Threshold-Free Variant

We vary the threshold in the binarized instantiation from Appendix E and compare it with direct use of r^{VLM} as the priority. We analyze an offline corpus of 2,000 32-frame MiniGrid clips collected from policies at six stages of training, from random to fully trained. Strict success records whether the goal predicate becomes true within the clip. Table 5 reports the raw-score distribution and balanced accuracy as τ varies.

Table 5: Offline threshold selection on the corpus. Bin share is the percentage of scores in each bin, indexed by its lower edge.

Edge τ	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65
Bin share (%)	1.45	13.95	10.30	33.05	11.30	10.50	16.00	3.20	0.25
Balanced accuracy	0.500	0.509	0.597	0.662	0.871	0.938	0.927	0.583	0.506

The raw-score distribution has modes near 0.40 and 0.55, and balanced accuracy peaks at the default threshold $\tau = 0.5$. We next test whether this offline sensitivity predicts downstream performance and whether thresholding is necessary. These VLM-only ablations use `DoorKey-16x16`/DQN; the thresholded arms vary τ in Equation 4, while the threshold-free arm uses r^{VLM} directly.

Table 6: Downstream VLM-only threshold ablation on `DoorKey-16x16`/DQN (3 seeds per configuration). The threshold-free row uses the raw score r^{VLM} directly as the VLM-only priority. ASR is measured at 1M steps; steps to 0.8 are averaged over solved runs. Uncertainty is SEM over seeds.

Score form	ASR@1M	Steps to 0.8
$\tau = 0.40$	0.53 ± 0.27	$755\mathrm{K} \pm 125\mathrm{K}$ (2/3)
$\tau = 0.45$	1.00 ± 0.00	$547\mathrm{K} \pm 135\mathrm{K}$
$\tau = 0.50$	1.00 ± 0.00	$357\mathrm{K} \pm 32\mathrm{K}$
Threshold-free (r^{VLM})	1.00 ± 0.00	$537\mathrm{K} \pm 143\mathrm{K}$

The thresholds with the highest offline balanced accuracy both reach 1.00 ASR, while $\tau = 0.40$ degrades performance. The threshold-free variant also reaches 1.00 ASR, but reaches 0.8 success

at $537\text{K} \pm 143\text{K}$ steps, compared with $357\text{K} \pm 32\text{K}$ for the default threshold $\tau = 0.5$. Thus offline calibration and direct use of the score provide two alternatives to selecting the threshold.

F.3 Mixing Schedule

Table 7: Ablation of mixing schedule ($\lambda_{\max}$) on `DoorKey-16x16` with VLM-only priority. **Performance** denotes the highest average success rate ($M = 5$ seeds). **Sample efficiency** tracks the steps required to reach the *baseline's* best performance.

		Improvement	
$\lambda_{\max}$	**Base.**	**Perf.** (↑)	**Sample Eff.** (↓)
Pure priority from start	**UER**	N/A (0.00/0.24)	N/A (1000K/916K)
	PER	N/A (0.00/0.80)	N/A (1000K/592K)
0.25	**UER**	+233.3% (0.80/0.24)	+40.6% (544K/916K)
	PER	+0.0% (0.80/0.80)	-0.3% (594K/592K)
0.50	**UER**	+316.7% (1.00/0.24)	+62.9% (340K/916K)
	PER	+25.0% (1.00/0.80)	+35.5% (382K/592K)
0.75	**UER**	+266.7% (0.88/0.24)	+57.0% (394K/916K)
	PER	+10.0% (0.88/0.80)	+17.2% (490K/592K)
1.00	**UER**	+316.7% (1.00/0.24)	+69.0% (284K/916K)
	PER	+25.0% (1.00/0.80)	+40.2% (354K/592K)

To understand the effect of the final mixing coefficient $\lambda_{\max}$, we conduct an ablation on the `MiniGrid/DoorKey-16x16` task (Fig.7). We fix $T_{\text{schedule}} = 5 \cdot 10^5$, ensuring that λ_t is annealed linearly from 0 (corresponding to purely uniform sampling) to $\lambda_{\max}$ over the first half of training (right panel). We sweep $\lambda_{\max} \in \{0.25, 0.5, 0.75, 1.0\}$ and include a **pure-priority-from-start** baseline, which disables the schedule and sets $\lambda_t = 1$ from the first update. We observe that larger $\lambda_{\max}$ values reach 50% success marginally earlier than smaller values. Both $\lambda_{\max} = 0.5$ and the scheduled $\lambda_{\max} = 1.0$ arm reach a final ASR of 1.00, while $\lambda_{\max} = 0.75$ reaches 0.88. In contrast, pure priority from the first update fails to solve the task, with success rates remaining near zero. This suggests that mixing uniform samples into replay during early training is important for effective learning in this setting. We fix $\lambda_{\max} = 0.5$ for all main experiments to preserve uniform coverage, without further per-task tuning.

To separate score granularity from the mixture, we additionally use raw-score priorities. VLM-only reaches ASR 1.00 ± 0.00 with the default schedule and 0.07 ± 0.07 with no mixture, while the raw-score product $r^{\text{VLM}}|\delta|$ with no mixture reaches 0.00 ± 0.00 (three seeds each). Together with Table 7's hard-score results—1.00 at $\lambda_{\max} = 0.5$ and 0.00 for pure priority from the start (five seeds)—changing score granularity does not rescue fully prioritized sampling. Here, no mixture sets $\lambda_t = 1$ throughout; the prioritized sampler retains only its 10^{-6} numerical floor and draws no uniform samples.

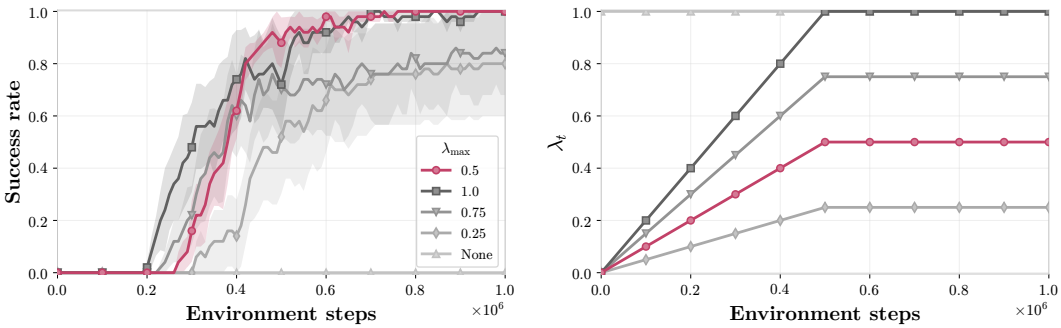


Figure 7: `MiniGrid/DoorKey-16x16`: $\lambda_{\max}$ ablation (5 seeds). "Pure priority from start" disables the warm-up and sets $\lambda_t = 1$ throughout. Left: performance for different $\lambda_{\max}$ values. Right: $\lambda_{\max}$ value evolution.

F.4 Realized Sampling Frequencies

At 250K steps in `Scene-5`/SAC, we count replay draws for each transition class and divide by the number expected under uniform replay. Table 8 reports the mean across five seeds using signal $\mathbf{p}^{\mathrm{VLM}}|\delta|$. The top-level frequencies have a class-size-weighted mean of 1.0. VLM-negative transitions constitute 81.1% of the buffer and are sampled at $0.91\times$ the uniform rate.

Table 8: Realized replay frequencies at the 250K buffer snapshot in `Scene-5`/SAC (five seeds). The successful-episode subset is contained within the VLM-positive row. A ratio of 1.0 is uniform replay, and 0.5 is the instantaneous mixture floor. Unscored transitions are newest, so their cumulative ratio may be lower because they have been present for fewer sampling steps.

Transition class	Buffer share	Frequency / uniform
VLM-positive	18.8%	$1.41\times$
Successful-episode subset	1.4%	$2.17\times$
VLM-negative	81.1%	$0.91\times$
No VLM score yet	0.1%	$0.44\times$

F.5 VLM Size

Table 9: Performance comparison of Perception-LM (Cho et al., 2025) models. Values in parentheses denote the relative percentage change compared to 1B. We use an Nvidia RTX 4090 GPU, and run 100 batches of 32 frames.

Model	Load (GiB)	Peak (GiB)	Time (s)	FPS
1B	2.86	3.77	0.46	69.27
3B	6.56 (+130%)	8.16 (+116%)	0.77 (+66%)	41.75 (-40%)
8B	18.25 (+539%)	20.34 (+439%)	2.15 (+366%)	14.88 (-79%)

We investigate how scaling the VLM affects both inference overhead and downstream RL performance. Table 9 quantifies the resource requirements of Perception-LM (Cho et al., 2025) variants for clip scoring (Nvidia RTX 4090 GPU; 100 batches of 32-frame clips). We observe that increasing the VLM size leads to a substantial increase in memory footprint and a corresponding reduction in throughput. Relative to the 1B model, the 3B variant increases peak memory by $+116\%$ and reduces FPS by 40%; the 8B variant increases peak memory by $+439\%$ and reduces FPS by 79%.

To assess the impact of VLM size on downstream performance, we compare the 1B, 3B, and 8B models on `MiniGrid/DoorKey-16x16` (Fig. 8). Notably, despite the increased inference cost of larger models, we do not observe consistent improvements in RL performance relative to the 1B configuration. These results suggest that in this setting, the clip-scoring signal saturates once the

VLM is sufficiently reliable at separating goal-consistent from goal-inconsistent segments. Given this trade-off, we select the 1B Perception-LM as the default backbone for all main experiments.

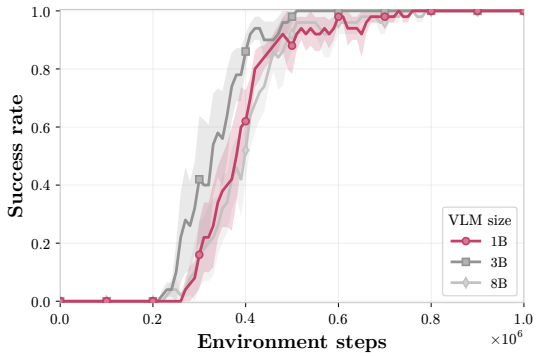


Figure 8: `MiniGrid/DoorKey-16x16`: VLM size ablation (5 seeds)

F.6 VLM Family Dependence

We score the offline corpus with three VLM families using the same prompt. Each threshold is selected on one half of the corpus and evaluated on the disjoint half. Table 10 reports held-out strict-success classification and full-corpus agreement with Perception-LM-1B.

Table 10: VLM-family diagnostics on the offline corpus. Balanced accuracy and AUC use the held-out half; agreement and raw-score Spearman ρ use the full corpus relative to Perception-LM-1B.

Model	τ	Bal. acc.	AUC	Agreement	ρ
Perception-LM-1B	0.50	0.94	0.98	100%	1.00
Gemma-4-E4B	0.12	0.88	0.92	86%	0.70
Qwen3-VL-2B	0.89	0.85	0.89	81%	0.73

The raw scores rank clips similarly across families, but their scales differ. We therefore compare offline threshold calibration with direct use of the raw score on `DoorKey-16x16`/DQN.

Table 11: VLM-only downstream comparison on `DoorKey-16x16`/DQN. The Perception-LM row contains 3 seeds; both Gemma rows contain 5. ASR is measured at 1M steps and uncertainty is SEM over seeds.

Model	f_{VLM}	Calibration	ASR@1M
Perception-LM-1B	Raw score	None	1.00 ± 0.00
Gemma-4-E4B	Raw score	None	0.80 ± 0.15
Gemma-4-E4B	$\mathbf{1}[r^{\text{VLM}} > 0.12]$	Offline threshold	1.00 ± 0.00

For Gemma, the calibrated threshold solves all five seeds; using the raw score requires no calibration and reaches ASR 0.80 ± 0.15. Thus a new VLM/prompt combination requires either a labeled calibration corpus for the hard-threshold variant or acceptance of the raw-score variant's observed performance trade-off.

F.7 Scorer Reliability

We evaluate the default Perception-LM-1B score against two simulator-derived labels on the offline corpus. Strict success records whether the goal predicate becomes true within the clip; milestone acquisition records key pickup, door opening, or the goal being reached.

Table 12: Reliability of the default binarized scorer at $\tau = 0.5$ on the offline corpus.

Ground-truth definition	Precision	Recall	FPR	FNR
Strict success	0.69	0.99	0.12	0.01
Milestone acquisition	0.92	0.62	0.04	0.38

Under strict success, the scorer produces 185 false positives and 3 false negatives. Of the false positives, 137 contain milestone acquisition and 29 more show positive net progress, leaving 19 clips with neither event (0.95% of the corpus and 3.2% of positive labels).

F.8 Robustness to Label Error

We test controlled label error on `DoorKey-16x16`/DQN using VLM-only priority and the binarized scorer. After thresholding, each positive clip label is demoted independently with probability κ, and the same number of negative clip labels are promoted uniformly without replacement. Unmatched demotions are carried to later scoring batches. The corruption changes training-time replay priorities only; rewards, bootstrap targets, and evaluation are unchanged.

Table 13: Controlled clip-label corruption on `DoorKey-16x16`/DQN with VLM-only priority (five seeds per configuration; mean ASR at 1M steps $\pm$ SEM). The deficit is the worst seed's terminal uncompensated demotions as a fraction of the cumulative positive-label count.

κ	0.00	0.10	0.25	0.50	0.75
ASR@1M	1.00 ± 0.00	0.96 ± 0.04	1.00 ± 0.00	0.40 ± 0.24	0.44 ± 0.20
vs. UER (0.20)	$+0.80$	$+0.76$	$+0.80$	$+0.20$	$+0.24$
Worst deficit	0.0%	0.0%	0.1%	4.0%	21.4%

Table 13 reports five seeds per corruption level, with $\kappa = 0$ reusing the uncorrupted VLM-only arm. ASR remains 0.96–1.00 through $\kappa = 0.25$, well above the measured 3.2% no-event label rate from Appendix F.7, and falls to 0.40–0.44 at the two higher levels. Paired substitution preserves the cumulative positive-label count within 0.1% for $\kappa \leq 0.25$; the higher-corruption points also reduce the prioritized set and therefore do not isolate precision alone.

F.9 Prompt Sensitivity

We re-score the offline corpus with the following four variants of the `MiniGrid/DoorKey` prompt.

1. **Paraphrase A.** *"Does this video clip visibly show the agent successfully completing a task goal or subgoal? Answer No when no clear success is visible, and do not infer hidden progress. Output exactly Answer: Yes or Answer: No."*
2. **Paraphrase B.** *"Is there clear visual evidence anywhere in this clip that the agent achieves an intended goal state? If the evidence is absent or ambiguous, answer No. Output exactly Answer: Yes or Answer: No."*
3. **Terse.** *"Is a task goal or subgoal visibly achieved in this clip? Output exactly Answer: Yes or Answer: No."*
4. **Underspecified.** *"Does anything important happen in this clip? Output exactly Answer: Yes or Answer: No."*

Table 14 reports fixed-threshold label agreement and raw-score rank correlation. The two semantic paraphrases preserve 90.9–95.3% of labels. The terse prompt labels every clip positive at $\tau = 0.5$, while the underspecified prompt labels every clip negative; their raw-score rankings remain correlated with the other prompts, and all five scores retain strict-success ROC AUC 0.955–0.989.

Table 14: Prompt sensitivity on the offline corpus. Lower-triangular entries are fixed-0.5 label agreement / raw-score Spearman correlation; the last column is strict-success ROC AUC.

Prompt	Base	Paraph. A	Paraph. B	Terse	Under.	AUC
Base	–					0.989
Paraphrase A	0.909 / 0.892	–				0.989
Paraphrase B	0.945 / 0.956	0.953 / 0.911	–			0.987
Terse	0.300 / 0.931	0.210 / 0.857	0.254 / 0.933	–		0.983
Underspecified	0.701 / 0.881	0.791 / 0.813	0.747 / 0.909	0.000 / 0.858	–	0.955

The most divergent semantic paraphrase, Paraphrase A, reaches ASR 0.84 ± 0.16 on `DoorKey-16x16` with VLM-only raw-score priority (five seeds).

F.10 LavaCrossing Analysis

We apply the analysis from Fig. 4 to `MiniGrid/LavaCrossing-S9N2`, where stepping into lava terminates the episode with zero reward. On a successful episode, the frozen VLM score crosses the threshold before the agent commits through the lava gaps, while the early critics remain flat ($|\Delta Q| \leq 0.03$); only the converged critic later develops the corresponding ΔQ surge (Fig. 9).

On a fatal episode, a large negative state-value difference does not appear. At the pre-terminal state, $Q(s, a_{\text{fatal}})$ reaches 0.109 at 55K training steps and falls to -0.007 at 200K, while $V(s) = \max_a Q(s, a)$ reaches 0.760. The learned hazard is action-conditional: stepping away remains viable, so a state-value curve cannot display the fatal action's value; the replay signal remains action-conditioned through its per-transition $|\delta|$.

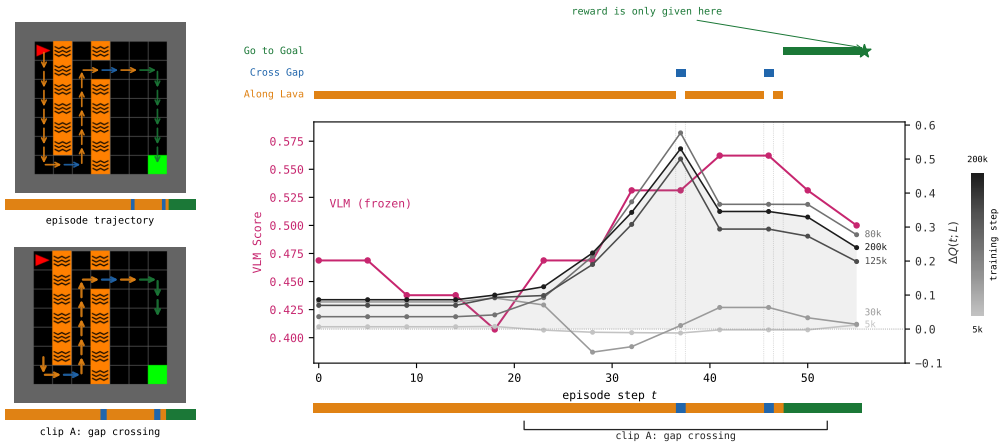


Figure 9: **Fig. 4 protocol on `LavaCrossing`.** On a successful episode, the frozen VLM score anticipates the learned value signal around the gap crossing.

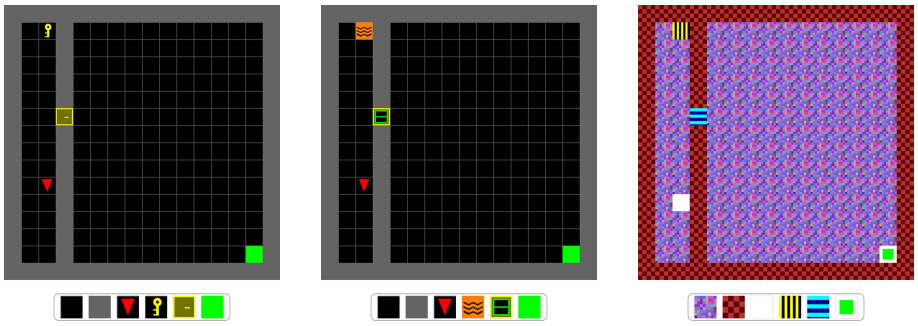
F.11 VLM Prior

Motivated by the "Modified Game" paradigm of Dubey et al. (2018), we probe whether **VLM-RB** leverages the VLM's pre-trained visual semantics to improve performance. To isolate the effect of visual semantics, we modify only the rendered frames provided to the VLM for scoring, while leaving both the underlying MDP and the agent's observations unchanged. If **VLM-RB** relies on semantic cues such as identifying keys, doors, or goal-relevant interactions, we expect its performance gains to diminish when these cues are distorted or removed.

To test this, we introduce two renderer perturbations (Fig. 10). First, `Sprite Swap` replaces object sprites with semantically conflicting alternatives, such as rendering keys as lava or doors as boxes,

thereby introducing misleading visual priors. Second, `Texture` replaces all object appearances with abstract high-contrast patterns, removing naturalistic semantics entirely.

Empirically, we observe that the unmodified setting achieves near-perfect success and converges the fastest (Fig. 5). In contrast, both `Sprite Swap` and `Texture` slow learning and reduce final success rates, with `Texture` also leading to the largest variance across seeds. Because the control problem and agent inputs remain fixed, this degradation suggests that the VLM produces less informative priorities when visual evidence for goal-relevant events is either misleading or lacks semantic content.



(a) Standard rendering (b) Semantically misleading sprites (c) Abstract patterns that remove naturalistic cues

Figure 10: Samples of the modified visuals. We modify only the frames passed to the VLM for scoring (agent observations and environment dynamics unchanged).

F.12 Computational Overhead and Scoring Lag

Table 15: **Training throughput** (steps/second) comparison on `MiniGrid/DoorKey-16x16`. Higher is better.

Hardware	PER	VLM-RB	Rel. Speed
NVIDIA A100	111	97	87%
NVIDIA A40	92	81	88%
NVIDIA A4000	76	67	88%

How much does **VLM-RB** actually slow down training in practice? To answer this, we measure throughput (steps per second) on the `MiniGrid/DoorKey-16x16` task using DQN, comparing **PER** and **VLM-RB** across three dual-GPU setups (NVIDIA A100, A40, and A4000). In each case, the RL learner and VLM are placed on separate devices. Notably, in this distributed configuration, the main bottleneck is inter-process communication (IPC) and data transfer, rather than competition for computational resources.

The results, summarized in Table 15, reveal a consistent and modest throughput reduction of about 12% across all hardware types. This suggests that, even when VLM inference is separated from the RL learner, the method maintains efficient scaling. In other words, the additional overhead introduced by **VLM-RB** remains limited in practical distributed settings.

Scoring lag. Across 20 logged `DoorKey-16x16` runs, the scorer keeps pace with collection: a backlog-based estimate gives zero median delay, the queues are empty at 99.7% of logged steps, the worst transient is 10.5K steps, and no frame is dropped.

It is important to note that these measurements use a standard inference setup, without any aggressive optimizations such as TensorRT or quantization (e.g., INT8 or FP4). We anticipate that employing a dedicated serving stack, such as vLLM or TGI, would further close the speed gap between **VLM-RB** and the baseline.

G EXPERIMENTS

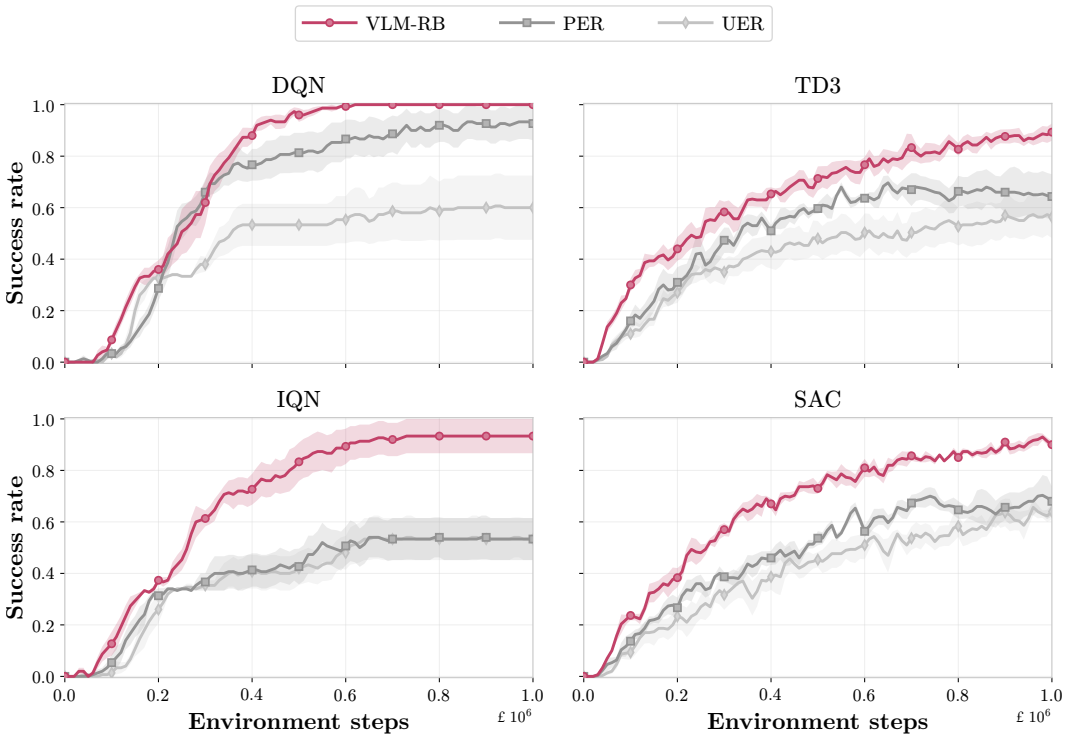
In this section, we present the full results of the baselines on all environments.

Table 16: **Performance** denotes the mean per-seed best success rate ($M = 5$ seeds). **Sample efficiency** tracks the steps required to reach the *baseline's* best performance; unreached targets are assigned the 1M-step horizon. **Wall-clock saving** estimates the reduction in real training time, accounting for a 12% inference overhead per step.

Alg.	Task	Baseline	Performance (↑) *Best ASR*	Sample Efficiency (↓) *Steps to Base. Best*	Wall-Clock Saving (↑) *Time vs. Baseline*
SAC	Scene-3	UER	+0.0% (1.00/1.00)	+50.7% (224K/454K)	+44.7%
		PER	+0.0% (1.00/1.00)	+18.8% (224K/276K)	+9.1%
	Scene-4	UER	+4.2% (1.00/0.96)	+54.2% (378K/826K)	+48.7%
		PER	+4.2% (1.00/0.96)	+52.8% (378K/800K)	+47.1%
	Scene-5	UER	+113.6% (0.94/0.44)	+60.0% (366K/914K)	+55.2%
		PER	+51.6% (0.94/0.62)	+34.7% (494K/756K)	+26.8%
TD3	Scene-3	UER	+0.0% (1.00/1.00)	+20.5% (202K/254K)	+10.9%
		PER	+0.0% (1.00/1.00)	+21.1% (202K/256K)	+11.6%
	Scene-4	UER	+47.1% (1.00/0.68)	+70.8% (212K/726K)	+67.3%
		PER	+0.0% (1.00/1.00)	+12.4% (410K/468K)	+1.9%
	Scene-5	UER	+200.0% (0.84/0.28)	+49.4% (366K/724K)	+43.4%
		PER	+90.9% (0.84/0.44)	+38.9% (522K/854K)	+31.5%
DQN	DoorKey-8x8	UER	+0.0% (1.00/1.00)	+14.3% (144K/168K)	+4.0%
		PER	+0.0% (1.00/1.00)	+30.8% (144K/208K)	+22.5%
	DoorKey-12x12	UER	+66.7% (1.00/0.60)	+55.4% (258K/578K)	+50.0%
		PER	+0.0% (1.00/1.00)	-9.9% (288K/262K)	-23.1%
	DoorKey-16x16	UER	+316.7% (1.00/0.24)	+64.2% (328K/916K)	+59.9%
		PER	+25.0% (1.00/0.80)	+40.2% (354K/592K)	+33.0%
IQN	DoorKey-8x8	UER	+0.0% (1.00/1.00)	+28.7% (134K/188K)	+20.2%
		PER	+0.0% (1.00/1.00)	+19.3% (134K/166K)	+9.6%
	DoorKey-12x12	UER	+56.2% (1.00/0.64)	+57.8% (296K/702K)	+52.8%
		PER	+56.2% (1.00/0.64)	+56.0% (296K/672K)	+50.7%
	DoorKey-16x16	UER	+250.0% (0.84/0.24)	+41.1% (512K/870K)	+34.1%
		PER	+425.0% (0.84/0.16)	+60.7% (256K/652K)	+56.0%

Table 17: **Performance** denotes the mean per-seed best success rate ($M = 5$ seeds). **Sample efficiency** tracks the steps required to reach the *baseline's* best performance. Both metrics are averaged across the aggregated algorithms and tasks.

Env Type	Agg. Algorithms	Baseline	Performance (↑) *Mean Best ASR*	Sample Efficiency (↓) *Mean Steps to Base Peak*
Scene	(SAC + TD3)	UER	+32.6% (0.96/0.73)	+55.2% (291K/650K)
		PER	+15.1% (0.96/0.84)	+34.6% (372K/568K)
DoorKey	(DQN + IQN)	UER	+57.0% (0.97/0.62)	+51.1% (279K/570K)
		PER	+27.0% (0.97/0.77)	+42.3% (245K/425K)



(a) MiniGrid/DoorKey (Grids 8,12,16) (b) OGBench/Scene (Tasks 3,4,5)

Figure 11: **VLM-RB consistently outperforms baselines across continuous and discrete tasks.** The plots show aggregated success rates for four algorithms (DQN, IQN, SAC, TD3) on MiniGrid and OGBench domains. Shaded regions indicate standard error across seeds.