SELF-TAUGHT POLICY IMPROVEMENT FOR VISION-LANGUAGE GAME AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models (VLMs) have achieved strong performance on visual question answering (QA), yet they are ineffective at playing games directly from raw pixels. This gap stems from a fundamental mismatch: VLM pre-training is based on image-grounded QA, whereas game playing requires goal-directed action generation, which is far outside the VLM pre-training distribution. Compared with raw action generation, in-game reasoning about the visual scene is structurally closer to image QA. Based on this insight, we propose a self-taught policy improvement framework that uses reasoning tokens, rather than action tokens, as the primary alignment target for VLM post-training in games, and requires no human annotation. Concretely, we first generate high-quality, state-grounded traces by rolling out the initial policy with privileged environment state information injected into the prompt, then distill these traces back into a pixel-only policy; iterating this rollout-and-distillation loop progressively improves reasoning and decision-making. Our approach raises Qwen3-VL-8B-Instruct's win rate from 0.0% to 31.3% on ProcGen-bigfish and from 13.8% to 58.1% on MiniHack, while more than doubling average Crafter achievements from 3.3 to 7.5. It also outperforms Gemini-3.5-Flash, SFT, GRPO, and GiGPO in task reward across all three games. Our approach outperforms all trained baselines in task reward with substantially fewer training samples.

1 INTRODUCTION

Vision-language models (VLMs) have achieved remarkable success in visual question answering (QA) and multi-modal understanding (Alayrac et al., 2022; Li et al., 2023; Achiam et al., 2023), yet as game agents, even state-of-the-art models often select invalid or suboptimal actions from raw pixels (Zhang et al., 2025a; Chen et al., 2024). The root cause is that VLM pre-training is centered on image QA, which requires perception and language generation rather than action generation (Zhang et al., 2026; Zitkovich et al., 2023; Kumar et al., 2024; Grigsby et al., 2025; Liu et al., 2025), placing sequential game decisions far outside the pre-training distribution.

Notably, in-game reasoning over the current state is structurally close to image QA, a task VLMs already excel at, and stronger reasoning is known to boost downstream performance (Yao et al., 2023; Shinn et al., 2023). This motivates shifting the post-training target from action tokens to reasoning tokens, turning the VLM's pre-trained strength into a bridge between visual QA and game play. However, most existing approaches do not exploit this insight. Supervised fine-tuning (SFT) methods imitate expert actions from large collections of state-action pairs (Tan et al., 2025; Bolton et al., 2025; Chen et al., 2025b), which requires costly annotations and often overfits demonstrations without grasping the game logic (de Haan et al., 2019; Baker et al., 2022). Reinforcement learning (RL) methods improve policies through environment interaction and reward-driven updates (Zhai et al., 2024; Wang et al., 2025b; Luo et al., 2025; Tong et al., 2026). When supervision is limited to sparse task outcomes, the learning signal does not directly identify which statements in a reasoning trace agree with the current environment. Recent approaches address this limitation by explicitly rewarding grounded reasoning (Wang et al., 2025a).

Self-taught learning, where a model improves from supervision generated by itself (Zelikman et al., 2022; Shenfeld et al., 2026), provides self-generated supervision for post-training and aligns naturally with a reasoning-centric view. By training on the model's own reasoning traces, it directly exercises the pre-trained language-side capability that VLMs already possess, without costly annotations or

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.

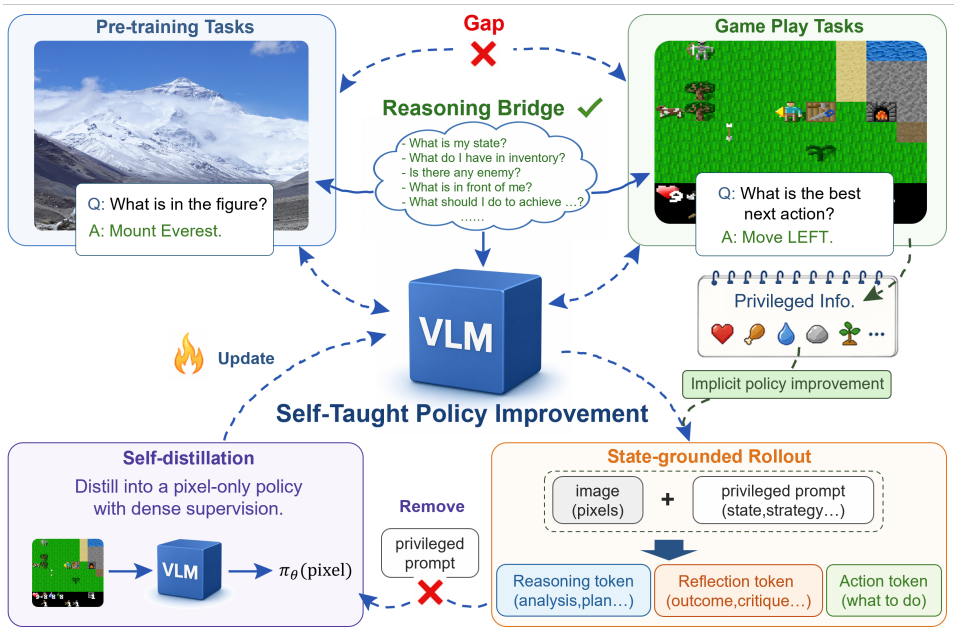


Figure 1: **Method overview.** VLMs excel at visual QA but are weak at action generation in games, leaving a gap between pre-training and game playing. In-game reasoning can bridge this gap. Our **Self-Taught Policy Improvement (STPI)** framework alternates between state-grounded rollout, where privileged prompts elicit high-quality reasoning, reflection, and action tokens, and self-distillation, where these traces are distilled into a pixel-only policy without human annotation.

environment rewards. However, existing self-taught methods are mostly language-only and rely on model-internal or outcome-level signals, leaving reasoning ungrounded in the environment. The key question we address is therefore *how to make self-taught learning viable for VLM agents*, which requires grounding the model's own reasoning in the actual environment without external annotations or hand-crafted reward. Many agentic settings, such as simulators and games, expose privileged information about the environment *at training time* that is hidden from the agent after training(Chen et al., 2020; Pinto et al., 2018). Based on this insight, we propose **STPI**, a **S**elf-**T**aught **P**olicy **I**mprovement framework to train VLM-based agents for sequential decision tasks in games (see Figure 1). During rollouts, STPI injects privileged information such as the game engine's state into the prompt so that the VLM produces grounded reasoning and actions. These high-quality, reasoning-rich trajectories are then distilled back into a pixel-only policy. STPI alternates between implicit policy improvement through privileged state conditioning and self-distillation that projects the retained reasoning–action traces into a pixel-only policy. Our experiments show that repeating this process improves reasoning and action quality.

We evaluate the proposed method on three pixel-only games of increasing complexity: ProcGen-bigfish (Cobbe et al., 2020), MiniHack (Samvelyan et al., 2021) and Crafter (Matthews et al., 2024). Across all three games, STPI achieves higher game rewards than Gemini-3.5-Flash and the trained baselines (SFT, GRPO (Shao et al., 2024), and GiGPO (Feng et al., 2025)). It outperforms SFT using the same number of training samples on Bigfish and MiniHack and 70% fewer on Crafter, while using approximately 79–93% fewer samples than the RL baselines and achieving lower reasoning hallucination rates than both (Table 1).

2 Related Work

Supervised Imitation for Vision-Language Game Agents. A dominant approach is SFT on expert state-action trajectories (Bolton et al., 2025; Chen et al., 2025b), e.g., Lumine (Tan et al., 2025), JARVIS-VLA (Li et al., 2025) and OmniJARVIS (Wang et al., 2024), which requires large-scale annotated data and curated pipelines and mainly copies behavioral patterns rather than the underlying game logic, generalizing poorly beyond data coverage. STPI removes the need for annotated trajectories by generating data through state-grounded rollouts.

Reinforcement Learning for Vision-Language Game Agents. RL methods such as RL4VLM (Zhai et al., 2024) and G1 (Chen et al., 2025a) optimize VLM policies using reward signals from environment interaction. VAGEN (Wang et al., 2025a) further grounds agent reasoning through rewards based on environment states. Related work on multimodal reasoning introduces step-wise reasoning rewards (Zhang et al., 2025b) and visually informed reward reweighting (Han et al., 2026). STPI instead uses privileged state to generate and filter reasoning–action traces, updating the policy through supervised self-distillation rather than task-reward optimization.

Modular Game Agents. Another line of work tackles complex game tasks through modular systems or harness design (Qin et al., 2024; Wang et al., 2023; Li et al., 2024), e.g. MP5 decomposes tasks into perception, planning, execution and verification modules (Qin et al., 2024); such approaches require significant system engineering and do not develop integrated reasoning within a single VLM. STPI instead improves reasoning, reflection and action in one end-to-end policy.

Self-Taught/Distilled Learning Paradigms. *Self-taught* learning constructs supervision from a model's own outputs, using selection or additional conditioning to improve the training signal. STaR (Zelikman et al., 2022) generates reasoning traces, uses known answers to regenerate rationales for incorrectly answered questions, and fine-tunes on traces that yield correct answers. Related self-distillation methods (Shenfeld et al., 2026; Zhao et al., 2026) condition the model on additional information, such as demonstrations or verified reasoning traces, to obtain teacher predictions for the same model operating without that context. Recent methods combine self-distillation with RL: SDPO (Hübotter et al., 2026) conditions a self-teacher on feedback and optimizes an on-policy logit-level distillation objective, while Self-Distilled RLVR (Yang et al., 2026) combines reward-directed updates with token credit assignment derived from privileged evidence. STPI uses privileged environment state both to elicit reasoning–action traces and to assess their grounding, retaining samples whose reasoning passes the grounding threshold and whose actions execute successfully. It then distills these self-generated traces through SFT into a policy that operates without privileged state.

3 METHODOLOGY

Our goal is to enable a VLM to play games from raw pixels through self-teaching, by closing the gap between image-grounded QA pre-training and goal-directed action generation. The key idea is to treat in-game reasoning as a bridge between pixels and actions, so that the VLM's pre-trained capability on language-side reasoning transfers to decision-making from visual inputs. Our framework first uses privileged environment state information during training to elicit high-quality reasoning traces, and then distills these traces into a policy that operates on vision-language inputs. By iterating state-grounded rollout and self-distillation, STPI uses the model's own grounded reasoning traces to improve reasoning and action selection from visual inputs.

3.1 PROBLEM SETUP

We model visual-input game play as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, \gamma)$ (Kaelbling et al., 1998), with latent state $s \in \mathcal{S}$, observation $o \in \mathcal{O}$, action $a \in \mathcal{A}$, and transition kernel $P(s_{t+1} \mid s_t, a_t)$. Each observation $o = (o^{\text{img}}, o^{\text{prompt}})$ pairs the current screenshot o^{img} with a textual prompt o^{prompt}, which may include a short action–outcome history, as in MiniHack. Privileged state is available only during training. The VLM policy $\pi_\theta(a_t \mid o_t)$ conditions on the current observation o_t; conditioning on the observation *history* (rather than only the current observation) is left for future work. The RL objective is the expected discounted return $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where R is the task reward and γ the discount factor. Here, T denotes the final permitted action index. Rollouts stop upon episode termination or exhaustion of the step budget; rewards after termination are taken to be zero.

3.2 OVERVIEW

Our method follows an iterative procedure $\pi_{\theta_{k+1}} = \mathcal{P}(\mathcal{I}(\pi_{\theta_k}))$, summarized in Algorithm 1: $\mathcal{I}$ constructs an augmented policy using privileged information (Sections 3.3–3.4) and $\mathcal{P}$ projects it back via self-distillation (Section 3.5).

Algorithm 1 State-Grounded Self-Taught Policy Improvement (STPI)

Require: Initial policy π_{θ_0}, environment $\mathcal{E}$, iteration number K

for $k = 0, 1, \ldots, K-1$ **do**
  // Implicit Policy Improvement via $\mathcal{I}$: $\pi^+_{\theta_k} \leftarrow \mathcal{I}(\pi_{\theta_k}) := \pi_{\theta_k}(a \mid \phi(o, s))$
  // State-Grounded Rollout
  $\mathcal{D}_k^{(1)} \leftarrow \emptyset$, $\mathcal{D}_k^{(2)} \leftarrow \emptyset$
  for each episode, initialize (o_0, s_0) and $t \leftarrow 0$ **do**
    while $t \leq T$ **do**
      Stage 1: $(w_t, a_t) \sim \pi^+_{\theta_k}(\cdot \mid o_t, s_t)$
      Execute a_t, observe z_t, o_{t+1}, s_{t+1}
      if $z_t =$ success **then**
        if $Q(w_t, s_t) \geq \tau$ **then** $\mathcal{D}_k^{(1)} \leftarrow \mathcal{D}_k^{(1)} \cup \{(o_t, w_t, a_t)\}$
        $t \leftarrow t + 1$
      else
        Stage 2: $(\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1}) \sim \pi^+_{\theta_k}(\cdot \mid o_{t+1}, s_{t+1}, w_t, a_t, z_t)$
        Execute $\hat{a}_{t+1}$, observe $\hat{z}_{t+1}$, o_{t+2}, s_{t+2}
        if $Q(\hat{w}_{t+1}, s_{t+1}) \geq \tau \ \wedge \ \hat{z}_{t+1} =$ success **then**
          $\mathcal{D}_k^{(2)} \leftarrow \mathcal{D}_k^{(2)} \cup \{(o_{t+1}, w_t, a_t, z_t, \tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1})\}$
        $t \leftarrow t + 2$
  $\mathcal{D}_k \leftarrow \mathcal{D}_k^{(1)} \cup \mathcal{D}_k^{(2)}$
  // Projection (Self-Distillation) via $\mathcal{P}$
  Strip privileged-state fields from the prompts in $\mathcal{D}_k$ to get $\tilde{\mathcal{D}}_k$
  Construct $\mathcal{D}^{(k)}_{\text{replay}} \leftarrow \bigcup_{j=0}^{k-1} \text{Sample}(\tilde{\mathcal{D}}_j, \ \alpha \cdot \beta^{k-j-1})$
  $\mathcal{D}^{\text{final}}_k \leftarrow \tilde{\mathcal{D}}_k \cup \mathcal{D}^{(k)}_{\text{replay}}$
  $\pi_{\theta_{k+1}} \leftarrow \arg\min_\pi \mathbb{E}_{(x,y)\sim\mathcal{D}^{\text{final}}_k}[-\log \pi(y \mid x)]$
return π_{θ_K}

Grounded-executability proxy for J. Rather than optimizing J through R as in RL, STPI pursues policy improvement via self-distillation over an augmented policy. To relate this procedure to the RL objective, we introduce a proxy objective $\tilde{J}$ based on per-step reasoning groundedness and action feasibility:

$$\tilde{J}(\pi) := \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \tilde{R}_t\right], \qquad \tilde{R}_t := \tilde{R}(w_t, s_t, z_t) := \lambda_1 Q(w_t, s_t) + \lambda_2 \mathbf{1}[z_t = \text{success}]. \tag{1}$$

where $Q(w_t, s_t) \in [0, 1]$ measures how faithfully the reasoning trace w_t matches s_t (detailed in Section 3.4), z_t is the environment outcome indicating whether the action a_t is successful, and $\lambda_1, \lambda_2 > 0$ are balance coefficients; While $\tilde{R}$ is used only for analysis, the practical rollout filter requires both $Q \geq \tau$ and successful action execution. We posit that improved reasoning groundedness and action executability support higher task returns, expressed through the working ordering assumption $\tilde{J}(\pi_1) \geq \tilde{J}(\pi_2) \Rightarrow J(\pi_1) \gtrsim J(\pi_2)$. The state-grounding and filtering ablations support this motivation: both components improve grounding and task reward (Table 3).

3.3 Implicit Policy Improvement via State Grounding

To mitigate partial observability, we augment the policy with privileged environment state and strategy guidance: $\pi^+_\theta(a \mid o, s) := \pi_\theta(a \mid \phi(o, s))$, where ϕ constructs the augmented prompt while keeping θ fixed. We expect this augmentation to improve task return, $J(\pi^+_\theta) \geq J(\pi_\theta)$, and call this *implicit policy improvement*. Section 4.2 evaluates this expectation by comparing game rewards with and without privileged information at the same checkpoints.

3.4 State-Grounded Rollout with Reasoning and Reflection

Our goal is to construct state-grounded training examples for self-distillation. We roll out the augmented policy π_θ^+ to generate reasoning and actions, invoke reflection after failures, and filter the resulting samples by reasoning groundedness and action execution success.

(1) Reasoning and Action. The model generates a reasoning–action pair $(w_t, a_t) \sim \pi_\theta^+(\cdot \mid o_t, s_t)$.

(2) Conditional Reflection and Correction. Executing a_t advances the environment to (o_{t+1}, s_{t+1}). If the action succeeds, the next decision starts normally from this new state. If it fails and the episode has not terminated or exhausted its step budget, the model reflects on the resulting frame and produces the next decision:

$$(\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1}) \sim \pi_\theta^+(\cdot \mid o_{t+1}, s_{t+1}, w_t, a_t, z_t), \tag{2}$$

where $\tilde{s}_{t+1}$ describes the current post-failure state, c_t explains why the previous action failed, and $(\hat{w}_{t+1}, \hat{a}_{t+1})$ are the corrected reasoning and action for the next environment step. Thus reflection is not a retry on o_t: the failed action has already consumed one step and changed the state.

Sample Filtering. To ensure reliable supervision, we filter samples from both stages jointly by reasoning quality and execution status. We prompt the VLM to parse its own w_t into JSON and compute the match rate $Q(w_t, s_t) \in [0, 1]$ against the true state, which is thresholded by τ; z_t is taken directly from the environment. Writing $E(w_t)$ for this parsed state and $H(s_t)$ for the corresponding simulator attributes over a predefined set $\mathcal{C}$, the match rate is $Q(w_t, s_t) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} q_c(E(w_t)_c, H(s_t)_c)$, where $q_c \in [0, 1]$ uses exact matching for categorical attributes and tolerance-based matching for numeric or spatial ones. We set $\tau = 0.7$ for experiments in this work.

With $x_t^{(1)} = o_t$, $y_t^{(1)} = (w_t, a_t)$, $x_{t+1}^{(2)} = (o_{t+1}, w_t, a_t, z_t)$ and $y_{t+1}^{(2)} = (\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1})$:

$$\begin{aligned} \mathcal{D}_k^{(1)} &= \{\, (x_t^{(1)}, y_t^{(1)}) \mid Q(w_t, s_t) \geq \tau,\ z_t = \text{success} \,\}, \\ \mathcal{D}_k^{(2)} &= \{\, (x_{t+1}^{(2)}, y_{t+1}^{(2)}) \mid z_t = \text{failure},\ Q(\hat{w}_{t+1}, s_{t+1}) \geq \tau,\ \hat{z}_{t+1} = \text{success} \,\}, \end{aligned} \tag{3}$$

and the full dataset is $\mathcal{D}_k = \mathcal{D}_k^{(1)} \cup \mathcal{D}_k^{(2)}$. Both stages instantiate the same grounded-executability filter $\mathcal{G} := \{Q \geq \tau \ \wedge \ \text{candidate action succeeds}\}$, using z_t in Stage 1 and $\hat{z}_{t+1}$ in Stage 2; the latter is additionally triggered only when $z_t = \text{failure}$. The Stage-1 filter applies separate acceptance criteria to the two components of $\tilde{R}_t$ in Eq. (1). Example samples are shown in Appendix A.1.

3.5 Self-Distillation via Projection

We construct the training dataset for each iteration by removing the privileged prompt from the two (x, y) formats above. The supervision in either format is dominated by reasoning and reflection tokens rather than the single action token.

To mitigate catastrophic forgetting and policy oscillation across iterations, we incorporate an experience replay mechanism. At iteration k, we construct a replay dataset $\mathcal{D}_{\text{replay}}^{(k)}$ by sampling from previous rollout data with exponential decay. The sampling ratio for data from iteration j ($j < k$) is given by $\rho_j^{(k)} = \alpha \cdot \beta^{k-j-1}$, where $\alpha \in (0, 1]$ controls the base sampling ratio and $\beta \in (0, 1]$ is the decay factor. The replay dataset is then $\mathcal{D}_{\text{replay}}^{(k)} = \bigcup_{j=0}^{k-1} \text{Sample}(\tilde{\mathcal{D}}_j, \rho_j^{(k)})$, where $\text{Sample}(\cdot, \rho)$ denotes uniform random sampling of a ρ-fraction of the dataset. The final training dataset combines the current iteration's projected data with the replay buffer: $\mathcal{D}_k^{\text{final}} = \tilde{\mathcal{D}}_k \cup \mathcal{D}_{\text{replay}}^{(k)}$, and the policy is updated via supervised fine-tuning (SFT) with the standard negative log-likelihood (NLL) loss over either stage's target sequence y given its non-privileged input x:

$$\theta' = \arg\min_\theta \mathbb{E}_{(x,y) \sim \mathcal{D}_k^{\text{final}}} \left[-\log \pi_\theta(y \mid x) \right]. \tag{4}$$

A policy-gradient view. Filtered SFT increases the likelihood of accepted reasoning–action traces. Its gradient has the binary-reward score-function form underlying REINFORCE (Williams, 1992),

with acceptance indicator $\mathbf{1}[\mathcal{G}]$ (Appendix A.2). Because rollouts come from the privileged policy, this is a reward-weighted supervised update rather than an on-policy gradient of $\tilde{J}$; the binary filter also differs from the additive proxy reward $\tilde{R}$.

Equivalently, SFT projects the retained target distribution onto the student policy. Let $p_k(x, y)$ be the empirical distribution of $\mathcal{D}_k^{\text{final}}$, including both stages and replay. Then Eq. (4) is equivalent to

$$\pi_{\theta'} = \arg\min_{\pi} \mathbb{E}_{x \sim p_k} \left[\mathrm{KL}(p_k(\cdot \mid x) \,\|\, \pi(\cdot \mid x)) \right]. \tag{5}$$

This projection transfers self-generated reasoning, reflection, and action targets into a policy without explicit privileged-state inputs. Iterating rollout, filtering, and projection supports self-improvement, with grounding and task-return gains evaluated in Section 4.

3.6 Inference

At test time, the policy operates solely from visual observations o_t and necessary basic prompts without privileged information. The inference follows a conditional two-stage strategy:

Stage 1: Reasoning and Action. The model generates $(w_t, a_t) \sim \pi_\theta(\cdot \mid o_t)$ and executes a_t.

Stage 2: Conditional Reflection. If a_t succeeds (i.e., the environment returns no failure feedback such as colliding with an obstacle, interacting with a non-interactable object, etc.), the agent proceeds directly to the next step of reasoning and action in the resulting state. Upon failure, the model invokes reflection on the resulting frame to produce $(\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1}) \sim \pi_\theta(\cdot \mid o_{t+1}, w_t, a_t, z_t)$, where $\tilde{s}_{t+1}$ describes the post-failure state, c_t is the reflection, and $(\hat{w}_{t+1}, \hat{a}_{t+1})$ are the corrected reasoning and action executed as the next step. Reflection is therefore not a retry: the failed action has already consumed a step and moved the environment to (o_{t+1}, s_{t+1}). This design saves computation on successful actions while enabling recovery from failures.

4 Experiments

We organize our empirical evaluation around four questions. **(Q1)** Does STPI improve the reasoning and decision-making quality of VLM agents? **(Q2)** Is STPI data-efficient compared with action-level supervision and game-reward-driven RL? **(Q3)** Does STPI distill the VLM's language-token reasoning into vision-conditioned generation, making grounded reasoning reachable from pixels alone? **(Q4)** How do the key components contribute to training stability and final performance?

4.1 Environments and Baselines

Environments. We evaluate our method on three pixel-only games of increasing complexity. In ProcGen-bigfish (Cobbe et al., 2020), the agent must eat 30 smaller fish while avoiding larger ones. MiniHack (Samvelyan et al., 2021) is a sandbox of short-horizon NetHack skills; we evaluate on ten tasks spanning crossing, levitation, traps, combat, and maze exploration. Crafter (Matthews et al., 2024) is an open-world 2D survival game where the agent must gather, craft, fight, and explore in a procedurally generated world while managing health and hunger. In every game the visual input is the current rendered frame and the policy selects from a game-specific discrete action set; MiniHack additionally provides a short textual action–outcome history. Privileged state is injected only during training rollouts, and evaluation seeds are disjoint from training seeds. Per-game observation and action spaces, evaluation protocols, and metric definitions are given in Appendix A.3.

Baselines. We compare STPI against five baselines spanning two categories. (1) *Zero-shot VLMs*: Qwen3-VL-8B-Instruct (Bai et al., 2025) and Gemini-3.5-flash (Google DeepMind, 2026), with the same basic prompt, action space, and failed-action feedback as STPI but without task-specific training. (2) *Trained agents*: SFT, for which rule-based scripts generate action data for ProcGen-bigfish and MiniHack while a PPO-RNN with superhuman performance generates expert state–action pairs on Crafter following Park et al. (2025); GRPO (Shao et al., 2024), a reinforcement learning baseline using only accumulated game reward; and GiGPO (Feng et al., 2025), a GRPO variant with step-level game reward. All trained baselines use the same Qwen3-VL-8B-Instruct backbone

Table 1: **Main results across ProcGen-bigfish, MiniHack, and Crafter**. Shared columns compare zero-shot, SFT, and RL baselines with STPI. The last two columns are STPI with action-token supervision only (**act.**) and with joint reasoning and action supervision (**Full**).

Env.	Metric	Qwen3VL	Gemini-3.5-flash	SFT	GRPO	GiGPO	STPI	
							act.	Full
Bigfish	Reas. Halluc. (↓)	48.3%	51.1%	/	39.8%	37.3%	/	**14.6%**
	Reward (↑)	0.56	3.00	5.41	7.96	7.30	5.38	**13.88**
	Winning Rate (↑)	0.0%	0.0%	0.0%	6.3%	0.0%	0.0%	**31.3%**
	Surv. Steps (↑)	82	80	156	144	139	174	**271**
	Train Data	0	0	∼15k	∼120k	∼127k	∼15k	∼15k
MiniHack	Reas. Halluc. (↓)	37.2%	**24.5%**	/	34.1%	36.1%	/	26.2%
	Winning Rate (↑)	13.8%	50.0%	53.8%	36.9%	32.5%	38.1%	**58.1%**
	Reward (↑)	−0.16	0.41	0.30	0.23	0.22	0.26	**0.52**
	Act. Succ. (↑)	68.1%	89.3%	82.1%	84.3%	85.2%	91.4%	**94.1%**
	Train Data	0	0	∼18k	∼242k	∼230k	∼18k	∼18k
Crafter	Reas. Halluc. (↓)	49.61%	42.09%	/	41.64%	42.85%	/	**28.91%**
	Act. Succ. (↑)	30.48%	33.83%	46.42%	46.32%	49.21%	48.92%	**59.80%**
	Reward (↑)	12.1	14.8	16.9	16.3	20.5	15.2	**27.9**
	Achiev. (↑)	3.3	3.9	2.2	4.2	4.9	2.0	**7.5**
	Surv. Steps (↑)	122	103	158	142	149	143	**171**
	Best Material	wood	stone	stone	stone	stone	stone	**coal**
	Best Tool	wood pickaxe	wood pickaxe	wood pickaxe	wood pickaxe	wood pickaxe	wood pickaxe	**stone pickaxe**
	Best Weapon	/	/	/	/	/	/	**wood sword**
	Best Defeated Crt.	zombie	cow	zombie	cow	zombie	zombie	**skeleton**
	Train Data	0	0	∼100k	∼144k	∼150k	∼30k	∼30k

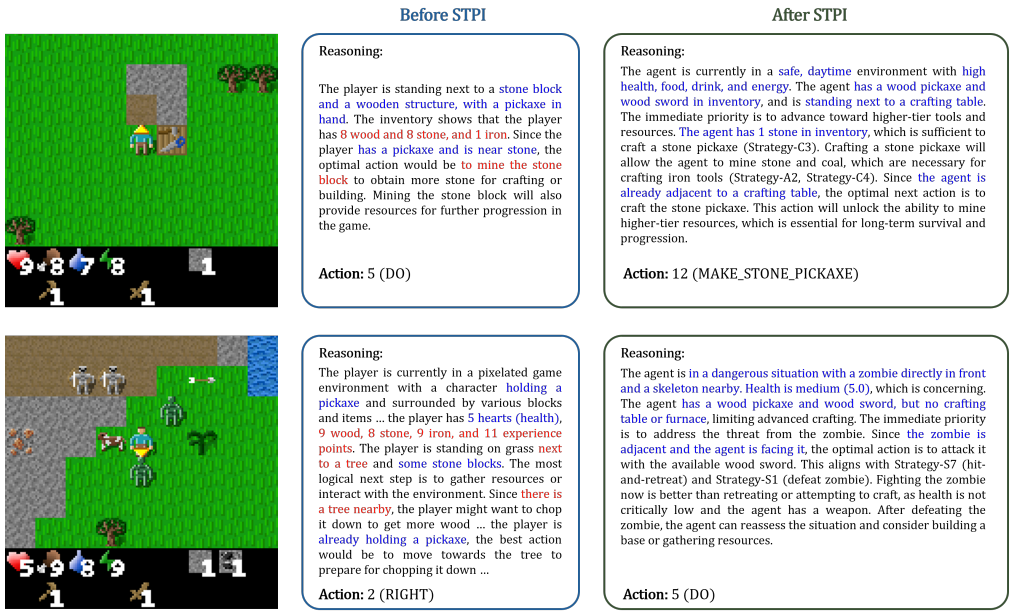


Figure 2: **Reasoning examples before and after STPI training on identical observations.** Each row shows one Crafter state (left) together with the pixel-only reasoning and action produced by Qwen3-VL-8B-Ins before (middle) and after STPI (right). Blue text marks reasoning content that is consistent with the true state, while red text marks hallucinations.

as STPI. GRPO and GiGPO use the same inference protocol as STPI, including the failed-action feedback of Section 3.6; SFT omits it because its demonstrations contain no environment outcomes. Implementation details and hyper-parameters are provided in Appendix A.4.

4.2 Main Results

Table 1 compares STPI with all baselines; we draw three main observations.

STPI improves both game play and reasoning. Full STPI attains the highest reward and game progress on all three games: it is the only method that wins bigfish episodes consistently (31.3% vs.

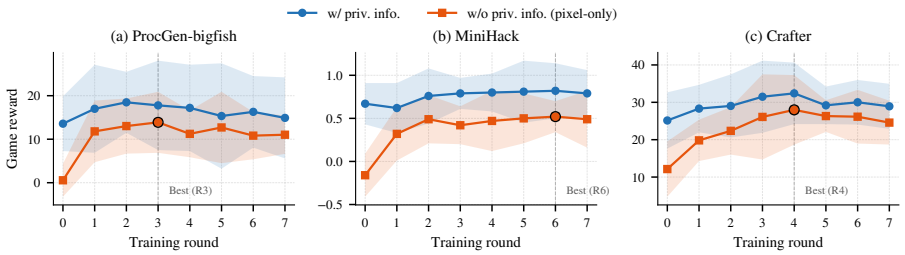


Figure 3: **Average game reward ($\pm$1 std.) across training rounds for full STPI.** Panels follow increasing complexity: (a) ProcGen-bigfish, (b) MiniHack, (c) Crafter. Blue: evaluation with privileged information; orange-red: pixel-only evaluation. The circled point marks the round with the highest pixel-only reward, whose checkpoint is reported in Table 1.

at most 6.3%), achieves the highest winning rate and action success on MiniHack, and on Crafter unlocks the most achievements while being the only agent that mines coal, crafts a stone pickaxe and a wood sword, and defeats skeletons. These gains come with lower reasoning hallucination than every rl baseline sharing the same backbone (e.g., 14.6% vs. 37.3% on bigfish and 28.9% vs. 41.6% on Crafter).

Reasoning bridges pixels and game play. The **act.** variant distills the same privileged rollouts but supervises only the action tokens. Without reasoning, the gain largely vanishes: action-only STPI performs on par with SFT on bigfish and Crafter (reward 5.38 vs. 5.41 and 15.2 vs. 16.9) and never wins a bigfish episode, while on MiniHack it reaches high action success (91.4%) yet a winning rate 20 points below Full, producing locally valid moves without goal-directed behavior. Privileged action labels alone therefore do not transfer to a pixel-only policy; it is the reasoning trace, which verbalizes what the frame shows and why an action follows, that allows the policy to recover privileged-quality decisions from pixels.

STPI is data-efficient. STPI uses 15k–30k samples, 5–13$\times$ fewer than GRPO and GiGPO, yet outperforms both on every game; with the same budget as SFT on bigfish and MiniHack, it still wins more often. We attribute this efficiency to dense supervision on reasoning tokens, which carries more learning signal per sample than action tokens or sparse rewards.

Figure 2 illustrates improved visual grounding and action selection after STPI on identical Crafter observations. For example, the pre-trained model proposes moving toward a tree despite an adjacent zombie, whereas the post-STPI model identifies the immediate threat and selects an attack. Figure 3 provides direct empirical evidence for implicit policy improvement (Section 3.3): privileged evaluation achieves higher average game reward than pixel-only evaluation at every round in all three games. Pixel-only reward improves beyond the first training round in all three games, supporting the benefit of iteration. Compared with initialization, the trained pixel-only policies achieve higher rewards and smaller gaps to their privileged counterparts in all three games, consistent with self-distillation transferring the benefits of privileged conditioning.Performance fluctuates across rounds; we use experience replay to mitigate forgetting.

4.3 Distilling Reasoning into Vision-Conditioned Generation

A central claim of STPI is that reasoning supervision transfers the VLM's *language-token* reasoning ability onto *vision-conditioned* outputs, so that the pixel-only policy behaves as if it had access to privileged information. We study this mechanism at the token level through a Crafter case study.

On 100 randomly selected testing states from Crafter, we first sample a reference reasoning trace $R^\star$ from the pre-STPI model with the pixel + privileged prompt, which reflects the grounded reasoning the VLM can already produce when privileged facts are given in the language prompt. We then score this *same* $R^\star$ as the target span under both the pre- and post-STPI models fed with a pixel-only prompt (teacher-forced, one forward pass), and report, averaged over $R^\star$, the per-token negative log-likelihood $\mathrm{NLL}(R^\star \mid o_t) = -\frac{1}{|R^\star|}\sum_k \log p_\theta(r_k \mid o_t, r_{<k})$ and its perplexity $\mathrm{PPL} = \exp(\mathrm{NLL})$. Lower NLL / PPL means the model assigns higher probability to the grounded trace, so $\Delta = \mathrm{Post} - \mathrm{Pre}$ quantifies how much easier the grounded reasoning becomes for the pixel-only model after STPI.

Table 3: **Ablation study on STPI components.** Δ is the change relative to full STPI in Table 1. *Progress* is winning rate for bigfish and MiniHack, and achievement count for Crafter.

Env.	Variant	Δ Reward $\uparrow$	Δ Halluc. $\downarrow$	Δ Act. Succ. $\uparrow$	Δ Progress $\uparrow$	Δ Surv. Steps $\uparrow$
Bigfish	w/o Reflection	-0.3	-2.6%	/	$+0.0\%$	-4.8
	w/o State-Grounding	-5.9	$+12.4\%$	/	-25.0%	-72.0
	w/o Filtering	-0.9	$+4.3\%$	/	$+0.0\%$	$+14.0$
	w/o Replay	-1.5	$+3.9\%$	/	-12.5%	-32.0
MiniHack	w/o Reflection	-0.04	-1.9%	-2.7%	-4.3%	/
	w/o State-Grounding	-0.23	$+9.9\%$	-6.8%	-18.1%	/
	w/o Filtering	-0.10	$+6.5\%$	-4.2%	-5.6%	/
	w/o Replay	-0.10	$+4.8\%$	-2.7%	-5.6%	/
Crafter	w/o Reflection	-6.8	$+5.0\%$	-8.4%	-2.5	-3.0
	w/o State-Grounding	-10.9	$+29.4\%$	-15.0%	-3.6	-38.0
	w/o Filtering	-2.6	$+4.2\%$	-2.0%	-0.4	-3.0
	w/o Replay	-4.0	-1.6%	-5.1%	-0.7	$+12.0$

Table 2 shows that STPI uniformly lowers the cost of generating the grounded trace $R^\star$ from pixels alone ($\Delta\mathrm{NLL}{=}-0.095$, median PPL ratio 0.91) across all four aggregates. Since both models see the same pixels and target, the drop directly measures how much more naturally the post-STPI model produces grounded reasoning that was previously reachable only with privileged inputs, providing token-level evidence that STPI routes the VLM's language-token reasoning ability into vision-conditioned generation—the mechanism behind the behavior-level gains in Table 1.

Table 2: **Token-level NLL/PPL of a privileged reference $R^\star$ from pixels. Pre/Post**: before/after STPI. **Ref.**†: privileged-prompt upper bound (not comparable). Δ = Post $-$ Pre (lower is better).

Metric	Pre	Post	Ref.†	Δ
mean NLL $\downarrow$	1.363	**1.272**	0.381	-0.091
median NLL $\downarrow$	1.366	**1.271**	0.381	-0.095
mean PPL $\downarrow$	4.02	**3.66**	1.47	-0.36
median PPL $\downarrow$	3.92	**3.56**	1.46	-0.36
PPL ratio (Post / Pre, median): **0.91**				

4.4 Ablation Analysis

We ablate four components of STPI: reflection, state grounding, reasoning filtering, and experience replay. Table 3 reports the change of each variant relative to full STPI.

Removing any component lowers reward in all three games, but the magnitude differs markedly. State grounding matters most: removing privileged state from rollouts causes the largest drops in reward and game progress and the largest increase in hallucination, demonstrating its central role in generating effective self-supervision. Reflection enables recovery from failed actions, and removing it in both training and inference lowers performance. Reasoning filtering keeps poorly grounded traces out of the distillation data, and dropping it raises hallucination in every game. Experience replay (Section 3.5) counteracts forgetting across iterations, and removing it reduces reward and game progress.

5 Conclusion and Discussion

STPI bridges QA-centric VLM pretraining and pixel-only game play through reasoning-centered self-teaching. It alternates privileged state-grounded rollouts with self-distillation of reasoning and reflection traces into a pixel-only policy without human annotation. Across three games, STPI achieves higher task rewards than zero-shot, SFT, and RL baselines, using fewer training samples than the RL baselines. It also improves action success and reduces reasoning hallucination relative to the trained baselines. Supervising actions alone on the same rollouts recovers little of this gain, highlighting the importance of reasoning supervision.

Several limitations still remain. First, STPI relies on privileged state information during training, which is available in simulators but may be hard to obtain in real-world tasks. Second, with single-frame visual observations the agent still falls short of human-level performance on long-horizon pixel-only game play. Third, policies are trained per game and do not yet transfer to unseen games, and iterative self-distillation can suffer mild forgetting in later rounds. These point to future directions: replacing privileged state with latent or world-model states, conditioning on visual histories with hierarchical planning, multi-game co-training with more stable iterative updates, and integrating STPI with RL by rewarding the reasoning process itself.

AI USE STATEMENT

We used generative AI tools to identify related literature, suggest the structure of the paper, draft and polish parts of the text, and assist in implementing parts of the experimental code. All AI-assisted code was tested by the authors, and all references and AI-assisted text were verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not raise any ethical concerns. All experiments are conducted in simulated game environments (ProcGen-bigfish, MiniHack, and the Craftax implementation of Crafter) using publicly available models or APIs. STPI requires no human annotation, and no human subjects, personally identifiable information, or private data are involved. We are not aware of any potential harmful applications or conflicts of interest arising from this work.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our results. The STPI procedure is specified in Section 3 and Algorithm 1, including the state-grounded rollout, the sample filtering rule with threshold τ, and the self-distillation objective. Appendix A.1 provides the full prompt templates, output examples, and SFT sample formats for all three games. Appendix A.2 states the simplifying assumptions and gives the complete derivation of the policy-gradient view. Appendix A.3 documents the observation and action spaces, step caps, reward definition (Eq. (12)), evaluation protocol on held-out seeds, and metric definitions for each environment. Appendix A.4 describes all baselines and lists the environment-specific settings and training hyper-parameters of every method (Tables 4 and 5). All trainable methods use the publicly available Qwen3-VL-8B-Instruct backbone, and all environments are built on public implementations.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, 2022.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, 2022.

Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. SIMA 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 66–75. PMLR, 2020.

Liang Chen, Hongcheng Gao, Tianyu Liu, Zhiqi Huang, Flood Sung, Xinyu Zhou, Yuxin Wu, and Baobao Chang. G1: Bootstrapping perception and reasoning abilities of vision-language model via reinforcement learning. *arXiv preprint arXiv:2505.13426*, 2025a.

Peng Chen, Pi Bu, Jun Song, Yuan Gao, and Bo Zheng. Can VLMs play action role-playing games? take black myth wukong as a study case. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Peng Chen, Pi Bu, Yingyao Wang, Xinyi Wang, Ziming Wang, Jie Guo, Yingxiu Zhao, Qi Zhu, Jun Song, Siran Yang, et al. Combatvla: An efficient vision-language-action model for combat tasks in 3D action role-playing games. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10919–10928, 2025b.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 2048–2056, 2020.

Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Google DeepMind. Gemini 3.5 Flash model card, 2026.

Jake Grigsby, Yuke Zhu, Michael Ryoo, and Juan Carlos Niebles. Vlm q-learning: Aligning vision-language models for interactive decision-making. *arXiv preprint arXiv:2505.03181*, 2025.

Yuhang Han, Yuyang Wu, Zhengbo Jiao, Yiyu Wang, Xuyang Liu, Shaobo Wang, Hanlin Xu, Xuming Hu, and Linfeng Zhang. Bridging visual representation and reinforcement learning from verifiable rewards in large vision-language models. *arXiv preprint arXiv:2603.27375*, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X.

Nishanth Kumar, William Shen, Fabio Ramos, Dieter Fox, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Caelan Reed Garrett. Open-world task and motion planning via vision-language model inferred constraints. *arXiv preprint arXiv:2411.08253*, 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.

Muyao Li, Zihao Wang, Kaichen He, Xiaojian Ma, and Yitao Liang. Jarvis-VLA: Post-training large-scale vision language models to play visual games with keyboards and mouse. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17878–17899, 2025.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Mingyu Liu, Zheng Huang, Xiaoyi Lin, Muzhi Zhu, Canyu Zhao, Zongze Du, Yating Wang, Haoyi Zhu, Hao Chen, and Chunhua Shen. Bridge thinking and acting: Unleashing physical potential of VLM with generalizable action expert. *arXiv preprint arXiv:2510.03896*, 2025.

Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. GUI-R1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2024.

Junyeong Park, Hyeonseo Cho, and Sungjin Ahn. CrafterDojo: A suite of foundation models for building open-ended embodied agents in Crafter. *arXiv preprint arXiv:2508.13530*, 2025.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018.

Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. MP5: A multi-modal open-ended embodied system in minecraft via active perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16307–16316. IEEE, 2024.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. MiniHack the planet: A sandbox for open-ended reinforcement learning research. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Weihao Tan, Xiangyang Li, Yunhao Fang, Heyuan Yao, Shi Yan, Hao Luo, Tenglong Ao, Huihui Li, Hongbin Ren, Bairen Yi, et al. Lumine: An open recipe for building generalist agents in 3d open worlds. *arXiv preprint arXiv:2511.08892*, 2025.

Jingqi Tong, Jixin Tang, Hangcheng Li, Yurong Mou, Ming Zhang, Jun Zhao, Yanbo Wen, Fan Song, Jiahao Zhan, Yuyang Lu, Chaoran Tao, Zhiyuan Guo, Jizhou Yu, Tianhao Cheng, Zhiheng Xi, Changhao Jiang, Zhangyue Yin, Yining Zheng, Weifeng Ge, Guanhua Chen, Tao Gui, Xipeng Qiu, Qi Zhang, and Xuanjing Huang. Game-RL: Synthesizing multimodal verifiable game data to boost VLMs' general reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, Qineng Wang, Hanyang Chen, Yiping Lu, Zhengyuan Yang, Lijuan Wang, Ranjay Krishna, Jiajun Wu, Li Fei-Fei, Yejin Choi, and Manling Li. VAGEN: Reinforcing world model reasoning for multi-turn VLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Zhengcheng Wang, Zichuan Lin, Yijun Yang, Haobo Fu, and Deheng Ye. SeeNav-Agent: Enhancing vision-language navigation with visual prompt and step-level policy optimization. *arXiv preprint arXiv:2512.02631*, 2025b.

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

Zihao Wang, Shaofei Cai, Zhancun Mu, Haowei Lin, Ceyao Zhang, Xuejie Liu, Qing Li, Anji Liu, Xiaojian Ma, and Yitao Liang. OmniJARVIS: Unified vision-language-action tokenization enables open-world instruction following agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025a.

Jianke Zhang, Xiaoyu Chen, Yanjiang Guo, Yucheng Hu, and Jianyu Chen. VLM4VLA: Revisiting vision-language-models in vision-language-action models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jingyi Zhang, Jiaxing Huang, Huanjin Yao, Shunyu Liu, Xikun Zhang, Shijian Lu, and Dacheng Tao. R1-VL: Learning to reason with multimodal large language models via step-wise group relative policy optimization. *arXiv preprint arXiv:2503.12937*, 2025b.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

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 Supplementary Material

This supplementary material provides details omitted from the main text. Appendix A.1 presents prompt templates, output examples, and SFT sample formats. Appendix A.2 motivates the grounded-executability proxy and derives the score-function interpretation and KL-projection formulation of filtered self-distillation. Appendix A.3 describes the environments, evaluation protocols, and metrics. Appendix A.4 summarizes training and inference settings.

A.1 Prompt Templates and Sample Formats

We describe the data constructed by the state-grounded rollout and consumed by the self-distillation SFT. A successful Stage-1 action may yield a sample (o_t, w_t, a_t). If that action fails, it first advances the environment to (o_{t+1}, s_{t+1}) and then triggers Stage 2 on the resulting frame; a successful correction may yield a sample with input (o_{t+1}, w_t, a_t, z_t) and target $(\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1})$. Both stages are converted into the same SFT JSON schema before training. We first show the Stage-1 prompt/output pair, then the Stage-2 pair, and finally the SFT dataset record, all for Crafter. Bigfish and MiniHack follow the same conditional two-stage structure; we list their prompts at the end of this section and show only the parts that differ.

Stage-1 prompt (reasoning and action). The Stage-1 prompt concatenates a role description, the observation specification, the 17-action space, a structured list of strategies identified by `Strategy-XN` codes, and the required output format. A condensed version is shown below (the full strategy list follows `Strategy-A1..A4`, `F1..F5`, `S1..S7`, `D0`, `M1..M12`, `P1..P4`, `C0..C7`, `R1`, `O1..O7`):

```
[image: game screenshot o_t]  +  [privileged state s_t appended as text,
↪  Stage-1 only]

You are a Crafter action decision assistant.
You control an agent in a top-down 2D world. You observe the world via an
image showing your local surroundings and inventory. You must maintain
health, food, water, and rest; episodes end on death or after 10,000
↪  steps.

Action Space (output the integer code):
 0: NOOP   1: LEFT   2: RIGHT   3: UP   4: DOWN
 5: DO     6: SLEEP  7: PLACE_STONE    8: PLACE_TABLE   9: PLACE_FURNACE
10: PLACE_PLANT       11: MAKE_WOOD_PICKAXE   12: MAKE_STONE_PICKAXE
13: MAKE_IRON_PICKAXE 14: MAKE_WOOD_SWORD     15: MAKE_STONE_SWORD
16: MAKE_IRON_SWORD

Basic strategies:
### Achievement dependency
(Strategy-A1) Wood path: Collect Wood -> Place Table -> Make Wood Pickaxe
↪  / Sword.
(Strategy-A2) Stone path: Collect Stone -> Place Stone -> Make Stone
↪  Pickaxe / Sword.
(Strategy-A3) Smelting and iron path: Place Furnace -> Collect Coal/Iron
↪  ->
              smelt iron -> Make Iron Pickaxe / Sword.
(Strategy-A4) Collect Diamond requires Iron Pickaxe.

### Food / drink / energy
(Strategy-F1) Eat Cow.   (Strategy-F2) Collect Sapling -> Place Plant ->
↪  Eat Plant.
(Strategy-F3) Collect Drink.   (Strategy-F4) Sleep restores energy.
(Strategy-F5) Wake Up after safe sleep.

### Combat and safety
(Strategy-S1) Defeat Zombie / Skeleton when necessary.
(Strategy-S2) Weapon priority: Iron > Stone > Wood > Bare hands.
(Strategy-S3) Retreat if health is low. (Strategy-S4) Avoid multi-enemy
↪  fights.
(Strategy-S5) Seek cover against ranged foes. (Strategy-S6)
↪  Hit-and-retreat.
(Strategy-S7) Engage if health is adequate and an enemy is near.

### DO
(Strategy-D0) Must face and be adjacent to the target; otherwise DO
↪  fails.

### Collecting materials
(Strategy-M1..M12) wood / stone / coal / iron / diamond / table / furnace
↪  /
drink / sapling / plant / wheat / meat, each with its required tool or
↪  action.

### Crafting and placement
(Strategy-P1..P4) Place Table (2 wood) / Furnace (1 stone) / Stone (1
↪  stone) /
                  Plant (1 sapling, on grass).
(Strategy-C0) Wood/stone tools need a table nearby.
(Strategy-C1) Iron tools need both a table and a furnace.
(Strategy-C2..C7) wood/stone/iron pickaxe and sword recipes.

### Reflection
(Strategy-R1) If your last action failed, try a DIFFERENT action this
↪  time.

### Others
```

```
(Strategy-O1..O7) base-building, earliest unmet prerequisite, tier
↪  progress,
smelting prerequisites, food/drink stability, decision checklist, and
default behavior when no immediate danger.

[Privileged state appended during training only]
True state s_t = { player: {pos, health, food, drink, energy, direction,
↪  ...},
                   inventory: {...}, nearby entities: [...],
                   affordances: {can_make_wood_pickaxe, can_sleep, ...} }

Output format:
- <explanation>explanation for why this action is optimal</explanation>
- <action>action_id (ACTION_NAME)</action>
- Do not output any extra text.
```

Stage-1 model output (example). The model emits one `<explanation>` span followed by one `<action>` span; the former is the reasoning trace w_t and the latter carries the action a_t:

```
<explanation>                                          # = w_t
The agent has 2 wood in inventory but no crafting table. Strategy-O2 says
↪  to
prioritize the earliest unmet prerequisite that unlocks further crafting,
↪  so
placing a table is the best next step. There is no immediate threat
↪  (health
8/8, no adjacent enemy), and the agent is standing on grass, satisfying
↪  the
placement condition (Strategy-P1). Placing the table enables Make Wood
Pickaxe next, which starts the Stone path (Strategy-A2).
</explanation>
<action>8 (PLACE_TABLE)</action>                # = a_t
```

Stage-2 prompt (reflection and correction). Stage 2 is invoked only after the Stage-1 action fails. Because the failed action has already consumed a step, the prompt uses the resulting frame o_{t+1} together with the previous `<explanation>`/`<action>` span and failure marker, and asks the model for four structured fields. The template is:

```
[image: resulting screenshot o_{t+1}]  +  [privileged s_{t+1} appended,
↪  Stage-2 only]

You are a Crafter action decision assistant performing self-reflection
↪  and
improvement. A Stage-1 model has already produced:

<explanation>{w_t from Stage-1}</explanation>
<action>{a_t from Stage-1}</action>

**Important: The above action FAILED when executed in the game.**

The failed action has already advanced the environment to the CURRENT
↪  image.
Given this resulting image and the Stage-1 output above, perform:
1. Describe the current post-failure state.
   Wrap in <outcome>...</outcome>.                    # = state
   ↪  description s~_{t+1}
2. Reflect on the Stage-1 reasoning: evaluate its accuracy and identify
↪  any
   issues. Wrap in <reflection>...</reflection>.      # = c_t
3. Produce the reasoning for the NEXT action based on your reflection.
   Wrap in <reasoning>...</reasoning>.                # = hat{w}_{t+1}
4. Output the next action consistent with the improved reasoning.
   Wrap in <action>action_id (ACTION_NAME)</action>.  # = hat{a}_{t+1}
```

Stage-2 model output (example). Below is a representative output, with the mapping to $(\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1})$ marked inline:

```
<outcome>                                          # = s~_{t+1}
The action MAKE_STONE_SWORD failed: the agent already has a stone sword
(inventory.stone_sword = 1) and can_make_stone_sword is false. Inventory
↪  and
map are unchanged; however health dropped from 8 to 6, indicating the
↪  zombie
in front continued to deal damage while the crafting attempt was
↪  rejected.
</outcome>
<reflection>                                       # = c_t
The Stage-1 reasoning ignored the affordance can_make_stone_sword = false
and the fact that a stone sword was already in inventory, so the chosen
crafting action was infeasible. The more urgent issue (an adjacent zombie
with health dropping) was not addressed.
</reflection>
<reasoning>                                        # = hat{w}_{t+1}
A zombie is directly in front and adjacent (Strategy-D0 satisfied), and
↪  the
agent already holds a stone sword (Strategy-S2). With health still 6
(adequate by Strategy-S7), engaging the zombie with DO is the correct
↪  choice:
it clears the immediate threat and avoids further health loss, whereas
↪  any
movement or crafting in this tile would waste a step under attack.
</reasoning>
<action>5 (DO)</action>                            # = hat{a}_{t+1}
```

SFT training sample format. After removing the privileged prompt, both Stage-1 and Stage-2 rollouts are serialized into the same JSON schema: a `conversations` list with a single human/gpt turn and an `images` list. During projection the supervision token span consists of the whole `gpt` turn (reasoning plus action for Stage-1; outcome, reflection, reasoning and action for Stage-2), which makes reasoning tokens dominate the loss.

A Stage-1 sample:

```
{
  "conversations": [
    {"from": "human", "value": "<image>\nYou are a Crafter action
    ↪  decision
        assistant.\n You receive the current observation image.\n
        Output the optimal next action for the current state.\n
        The action must be consistent with the explanation.\n
        Action space:[17-action space]...\n
        Output format requirement:\n
        - First output one explanation block:
        <explanation>reasoning for why this action is
        ↪  optimal</explanation>\n
        - Then output one action block: <action>action_id
        ↪  (ACTION_NAME)</action>\n
        - Do not output any extra text."},
    {"from": "gpt",   "value": "<explanation>The agent has 2 wood in
    ↪  inventory
        but no crafting table ... Placing the table enables Make Wood
        Pickaxe next.</explanation>\n<action>8 (PLACE_TABLE)</action>"}
  ],
  "images": [".../episode_00010_step_00233.png"]
}
```

A Stage-2 sample:

```
{
  "conversations": [
```

```
{"from": "human", "value": "<image>\nYou are a Crafter action decision
        assistant performing self-reflection and improvement.\n
        A stage-1 model analyzed the previous observation image
        and produced the following
        ↪  output:\n\n<explanation>...</explanation>\n
        <action>15 (MAKE_STONE_SWORD)</action>\n\n**Important: The above
        ↪  action
        FAILED and the displayed image is the resulting new
        ↪  frame.**\n\nNow ...
        1. <outcome>...</outcome>
        2. <reflection>...</reflection>  3. <reasoning>...</reasoning>
        4. <action>action_id (ACTION_NAME)</action>."},
    {"from": "gpt",   "value": "<outcome>The action MAKE_STONE_SWORD
    ↪  failed
        ...</outcome>\n<reflection>The stage-1 reasoning ignored the
        affordance ...</reflection>\n<reasoning>A zombie is directly in
        front ...</reasoning>\n<action>5 (DO)</action>"}
  ],
  "images": [".../episode_09321_step_00270.png"]
}
```

ProcGen-bigfish prompt templates. Bigfish follows the same two-stage schema with a compact JSON output instead of tagged blocks. The Stage-1 pixel-only prompt is shown below in condensed form; the privileged prompt is identical except that it also documents the ground-truth block appended to the user turn.

```
[image: game frame o_t]  +  [privileged block appended as text, training
↪  rollouts only]

You play Procgen `bigfish` (top-down 2D underwater) from the RGB frame
↪  ONLY.
Identify everything by COLOR, SHAPE and RELATIVE SIZE. EAT fish strictly
SMALLER than yours (+1; +10 for 30 eaten). Touching a LARGER fish ends
↪  the
episode (0 reward).

## Identifying YOUR fish
YOUR fish is ALWAYS the GREEN clownfish -- it is the ONLY green fish on
screen. A fish is "smaller" only if its bounding box is visibly (>=10%)
smaller than yours; treat ties as NOT edible.

## Actions
  0 UP     3 RIGHT    6 UP_RIGHT
  1 DOWN   4 NOOP     7 DOWN_LEFT
  2 LEFT   5 UP_LEFT  8 DOWN_RIGHT

## Threat tier for the NEAREST LARGER fish
IMMINENT   : same row/column band OR sprites almost touching.
CAUTIONARY : diagonal quadrant, roughly 3-7 body-lengths away.
NEGLIGIBLE : diagonal AND >~7 body-lengths away, or no larger fish.

## Priority (stop at first match)
  1. LARGER IMMINENT -> move in the direction that opens the gap fastest.
  2. Smaller fish visible -> move TOWARD the closest one.
  3. No smaller fish visible -> move toward the SCREEN CENTER.

## Output -- STRICT
ONE LINE of compact JSON, no markdown:
  {"reason": "<...>", "action": <int 0-8>}
```

The Stage-2 suffix used after a failed action:

```
The previous action already executed and FAILED (<reason>). The
↪  environment
has moved to a NEW frame.
```

```
Previous reason: <w_t>
Previous action: <a_t>
Describe the current state, write a brief reflection, then a new reason
and action.
ONE LINE JSON: {"outcome":"<...>", "reflection":"<...>",
↪  "reason":"<...>",
               "action": <int 0-8>}
```

MiniHack prompt templates. MiniHack reuses the tagged output format of Crafter. Because its action set is task-dependent, the available actions are listed in the prompt at every step, and the shared rules are followed by one family-specific strategy block selected by task type (`Z1..Z3` for mazes, `H1..H3` for hazards, `B1..B3` for horde battles, `K1` for dark rooms, `U1..U3` for item use).

```
[image: 9x9 tile crop o_t]  +  [state description appended, training
↪  rollouts only]

You are playing MiniHack (a NetHack dungeon task). The image is a
↪  top-down
tile view of the dungeon centred on your character. Black areas are
unexplored or out of sight. You may see stone walls, room floor,
↪  corridors,
doors, a staircase down, items and monsters.

Rules of movement and interaction:
- Compass moves (N/S/E/W and diagonals) move one tile. Moving into a
↪  monster
  ATTACKS it. Moving into a wall wastes the turn.
- Diagonal moves are ILLEGAL inside corridors and through doorways.
- Walking into a closed door OPENS it. Where the action list offers them:
  PICKUP, OPEN, KICK, EAT / WEAR / QUAFF / ZAP use inventory items. Only
  ↪  ever
  choose actions that appear in the available-actions list.
- You die at 0 HP; the episode also ends when the task is complete (often
↪  by
  reaching the staircase DOWN).

Basic strategies:
(Strategy-G1) If the goal is visible, move toward it.
(Strategy-G2) Otherwise explore unexplored dark areas systematically.
(Strategy-G3) Black tiles are UNEXPLORED, not walls.
(Strategy-G4) Do not bounce back and forth between the same tiles.
(Strategy-C1) Attack adjacent monsters only when your HP is comfortable.
(Strategy-C2) NEVER step into lava; cross hazards only while levitating.
(Strategy-D1) Closed door in the way: walk into it to open it.
(Strategy-I1) Stand on an item, then PICKUP; use task items when
↪  required.
(Strategy-R1) If your last action failed, do NOT repeat it.
(Strategy-O1) Checklist: adjacent threat? -> goal visible? -> nearest
              unexplored direction? -> then act.
[+ one family block, e.g. (Strategy-U1..U3) for item-use tasks]

Task: <task hint>
Available actions (pick exactly one): N, E, S, W, NE, SE, SW, NW, PICKUP,
↪  ...

Reply ONLY with this exact format:
<reasoning>your brief reasoning in 1-3 sentences</reasoning>
<action>ACTION_NAME</action>
```

The user turn carries the five-step action history and, in privileged rollouts, a natural-language rendering of the engine state. The privileged system prompt adds the two paragraphs below; the second keeps the teacher's reasoning phrased as a player would phrase it, so that the distilled traces remain reachable from pixels.

```
Recent actions (most recent first):
```

```
- 1 step ago: E (failed) [blocked_by_wall]
- 2 step ago: SE (ok)
...
Look at the current view and decide your next action.

[privileged only] In addition to the image, you receive a state JSON
describing the current dungeon view, your vitals, inventory and the last
game message. Use both to choose the best action.

[privileged only] STRICT REASONING CONSTRAINT: your <reasoning> text MUST
describe what you see in plain natural language. Do NOT mention any JSON
field name, do NOT quote raw numeric values from the state JSON. Refer to
objects the way a human player would (e.g. 'the staircase to my
↪  south-east').
```

The Stage-2 prompt used after a failed action keeps the same system prompt and action list, shows the resulting frame, and replaces the reply format:

```
[image: resulting 9x9 tile crop o_{t+1}]  +  [state description, training
↪  only]

Your previous action FAILED and the game has moved to the CURRENT view.
Previous reasoning: <w_t>
Previous action: <a_t> (failed) [<failure reason>]

Reply ONLY with this exact format:
<outcome>what the current view shows after the failure</outcome>
<reflection>why the previous action failed</reflection>
<reasoning>your brief reasoning for the next action</reasoning>
<action>ACTION_NAME</action>
```

A.2 Proxy Motivation and Policy-Gradient View of STPI

We motivate the proxy objective in Section 3.2, derive the gradient of filtered self-distillation, and establish its KL-projection interpretation. The derivation distinguishes the fixed-buffer supervised update used by STPI from an on-policy REINFORCE update.

A.2.1 Grounded-executability proxy. Recall $\tilde{R}_t = \lambda_1 Q(w_t, s_t) + \lambda_2 \mathbf{1}[z_t = \text{success}]$. Grounded reasoning supplies state-consistent information for action selection, while successful execution indicates that the proposed action is feasible. Together, these signals identify useful reasoning–action examples at individual decision steps, without relying on a terminal task reward to select supervision.

This motivates the working ordering assumption $\tilde{J}(\pi_1) \geq \tilde{J}(\pi_2) \Rightarrow J(\pi_1) \gtrsim J(\pi_2)$ in Section 3.2. The state-grounding and filtering ablations support this motivation: removing either component reduces task reward and increases reasoning hallucination (Table 3). The ordering remains an assumption about the relationship between the proxy and task return, rather than a consequence of the likelihood objective.

The practical filter applies the two criteria jointly: $g = \mathbf{1}[Q \geq \tau]\mathbf{1}[z = \text{success}]$. It is distinct from the additive proxy reward $\tilde{R}$. The coefficients λ_1, λ_2 therefore enter only the analysis; sample selection does not depend on them. On correction steps, the proxy and filter use $(\hat{w}_{t+1}, s_{t+1}, \hat{z}_{t+1})$. Both task and proxy rewards are zero after episode termination.

A.2.2 Exact gradient on a fixed rollout buffer. At iteration k, let q_k be the empirical distribution of candidate rollout records before filtering. Each record b contains a student input x_b, target sequence y_b, and the state and execution information needed to evaluate its acceptance indicator $g(b)$. For Stage 1, $(x_b, y_b) = (o_t, (w_t, a_t))$. For Stage 2, $x_b = (o_{t+1}, w_t, a_t, z_t)$ and $y_b = (\tilde{s}_{t+1}, c_t, \hat{w}_{t+1}, \hat{a}_{t+1})$; these candidates are generated only after a Stage-1 failure.

For acceptance rate $Z_k = \mathbb{E}_{b\sim q_k}[g(b)] > 0$, the current-round filtered distribution is $q_k^{\mathcal{G}}(b) = q_k(b)g(b)/Z_k$. The filtered NLL and its gradient are therefore

$$\mathcal{L}_k(\theta) = -\mathbb{E}_{b\sim q_k^{\mathcal{G}}}[\log \pi_\theta(y_b \mid x_b)] = -\frac{1}{Z_k}\mathbb{E}_{b\sim q_k}[g(b)\log \pi_\theta(y_b \mid x_b)], \tag{6}$$

$$-\nabla_\theta \mathcal{L}_k(\theta) = \frac{1}{Z_k}\mathbb{E}_{b\sim q_k}[g(b)\nabla_\theta \log \pi_\theta(y_b \mid x_b)]. \tag{7}$$

This identity is exact for the fixed empirical buffer: q_k, g, and Z_k are held constant during the update. It expresses filtered SFT as normalized binary-weighted maximum likelihood, with positive weight assigned to accepted targets.

A.2.3 Connection to REINFORCE. For comparison, consider a one-step objective with fixed input distribution $d(x)$ and fixed acceptance probability $r_g(x, y) \in [0, 1]$:

$$J_g(\theta) = \mathbb{E}_{x\sim d,\, y\sim\pi_\theta(\cdot|x)}[r_g(x, y)]. \tag{8}$$

Here r_g averages the binary acceptance indicator over any latent-state and execution randomness under a fixed conditional law. The score-function identity gives

$$\nabla_\theta J_g(\theta) = \mathbb{E}_{x\sim d,\, y\sim\pi_\theta(\cdot|x)}[r_g(x, y)\nabla_\theta \log \pi_\theta(y \mid x)]. \tag{9}$$

Equations (7) and (9) share the reward-weighted score-function form underlying REINFORCE (Williams, 1992), but use different sampling distributions. STPI generates its buffer with privileged conditioning and trains the student without that explicit conditioning. Consequently, Eq. (7) describes the actual supervised update, not an on-policy gradient of J_g or $\tilde{J}$. Moreover, J_g measures binary acceptance, whereas $\tilde{J}$ accumulates the additive proxy reward.

A.2.4 KL projection and replay. Let $p_k(x, y)$ denote the empirical distribution of $\mathcal{D}_k^{\text{final}}$, including accepted examples from both stages and sampled replay data. For this fixed distribution, the SFT loss decomposes as

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim p_k}[\log \pi_\theta(y \mid x)] \tag{10}$$

$$= \mathbb{E}_{x\sim p_k}\left[\mathrm{KL}(p_k(\cdot \mid x) \,\|\, \pi_\theta(\cdot \mid x))\right] + H_{p_k}(Y \mid X). \tag{11}$$

The conditional entropy $H_{p_k}(Y \mid X)$ is independent of θ. Thus minimizing NLL is exactly equivalent to the KL projection in Section 3.5, over the same parameterized policy family. Replay changes the empirical mixture p_k but preserves this equivalence; its loss contributes likelihood terms from earlier iterations.

The projection fits the retained reasoning, reflection, and action sequences under student inputs that omit explicit privileged state. The updated policy then generates the next round of privileged rollouts, completing the rollout–filter–projection loop. Figure 3 evaluates the reward advantage of privileged augmentation and the improvement of the pixel-only policy across rounds. These empirical comparisons complement the exact likelihood and projection identities above.

A.3 Environment Details

We describe the three games in order of increasing complexity. In all of them the policy receives only the current rendered frame and selects one action from a discrete, game-specific set; MiniHack additionally provides a short textual action–outcome history, but no past images. The privileged state is appended to the prompt as text during training rollouts only.

A.3.1 ProcGen-bigfish. The agent controls a fish that grows by eating strictly smaller fish and dies on contact with a larger one; a level is completed once 30 fish have been eaten, which yields $+1$ per fish and a $+10$ completion bonus. The observation is the native 512×512 RGB frame exposed by ProcGen, falling back to the standard 64×64 observation when the high-resolution buffer is unavailable, resized by nearest-neighbour interpolation; no textual history is added. The nine actions are:

ID	Action	ID	Action	ID	Action
0	UP	3	RIGHT	6	UP_RIGHT
1	DOWN	4	NOOP	7	DOWN_LEFT
2	LEFT	5	UP_LEFT	8	DOWN_RIGHT

The privileged block has six fields: the agent's size bucket, the direction and distance of the nearest smaller and of the nearest larger fish, the fastest approaching and escaping move, and a safe direction. Directions follow the game's axis-aligned bounding-box predicate, so that `left`/`right` (resp. `up`/`down`) means the two boxes already overlap on the other axis and a single step therefore eats or dies. We evaluate 16 episodes on held-out seeds, each capped at $1{,}500$ steps. *Winning rate* is the fraction of episodes reaching the 30-fish quota; an action counts as successful when it eats a fish or, for a non-`NOOP` move, changes the agent's position.

A.3.2 MiniHack. MiniHack exposes a large suite of short-horizon NetHack skills as separate tasks; we evaluate on ten of them. The agent sees only the 9×9 tile crop centred on its character, rendered at 144×144 and upscaled to 288×288, together with a five-step textual action history reporting whether each previous action succeeded. Unlike the other two games the action set is task-dependent: navigation tasks expose the eight compass moves; skill tasks add `PICKUP`, one task-specific item command (`QUAFF`, `ZAP` or `WEAR`) and `FIRE`, which answers the game's directional prompt; the maze task further adds `OPEN`, `KICK`, `SEARCH` and `EAT`. We use the following ten tasks:

Task	Actions	Max steps
WoD-Easy-Restricted	11	80
LavaCross-Levitate-Potion-Pickup-Restricted	11	160
SimpleCrossingS11N5	8	40
Levitate-Potion-Restricted	11	100
Freeze-Wand-Restricted	11	100
Levitate-Boots-Restricted	11	100
LavaCrossingS9N2	8	40
Room-Trap-5x5	8	40
Room-Monster-15x15	8	120
ExploreMaze-Hard-Mapped	12	200

Each step cap is $0.4\times$ the task's native budget clipped to $[40, 300]$, raised where a measured optimal solution would otherwise leave a fallible agent too little slack. The privileged prompt adds a natural-language rendering of the dungeon view, vitals, inventory and last game message. We evaluate 16 episodes per task on held-out seeds (160 episodes in total) and report the macro-average over tasks; an episode counts as a win when it terminates with a positive task reward.

A.3.3 Crafter. We build on the JAX-based Craftax re-implementation of Crafter, using its `craftax_classic` branch (`envs/Craftax/craftax/craftax_classic`). An episode is a procedurally generated 64×64 top-down 2D world in which the agent must manage health, food, drink and energy while progressing through a tool tech-tree. The episode terminates either when health reaches zero or after $10{,}000$ steps. We evaluate 16 episodes on held-out seeds. The agent selects one of the 17 discrete actions below at each step:

ID	Action	ID	Action
0	NOOP – do nothing	9	PLACE_FURNACE – place a furnace (1 stone)
1	LEFT – move left	10	PLACE_PLANT – plant a sapling on grass
2	RIGHT – move right	11	MAKE_WOOD_PICKAXE – at a table
3	UP – move up	12	MAKE_STONE_PICKAXE – at a table
4	DOWN – move down	13	MAKE_IRON_PICKAXE – at table+furnace
5	DO – interact with the front tile	14	MAKE_WOOD_SWORD – at a table
6	SLEEP – sleep to restore energy	15	MAKE_STONE_SWORD – at a table
7	PLACE_STONE – place a stone (1 stone)	16	MAKE_IRON_SWORD – at table+furnace
8	PLACE_TABLE – place a table (2 wood)		

DO is a context-sensitive action: depending on the tile in front of the agent and the tool held, it chops trees, mines stone/coal/iron/diamond, drinks water, collects saplings, eats cows, or attacks mobs. The same action index is therefore reused for *any* environment interaction.

A.3.4 Crafter achievements. The environment defines 22 achievements, enumerated in the order below. Each achievement carries a fixed difficulty weight:

ID	Achievement	Weight	ID	Achievement	Weight
0	COLLECT_WOOD	1	11	EAT_PLANT	2
1	PLACE_TABLE	2	12	DEFEAT_SKELETON	4
2	EAT_COW	2	13	MAKE_STONE_PICKAXE	4
3	COLLECT_SAPLING	1	14	MAKE_STONE_SWORD	4
4	COLLECT_DRINK	1	15	WAKE_UP	1
5	MAKE_WOOD_PICKAXE	3	16	PLACE_FURNACE	3
6	MAKE_WOOD_SWORD	3	17	COLLECT_COAL	2
7	PLACE_PLANT	1	18	COLLECT_IRON	3
8	DEFEAT_ZOMBIE	4	19	COLLECT_DIAMOND	5
9	COLLECT_STONE	1	20	MAKE_IRON_PICKAXE	4
10	PLACE_STONE	1	21	MAKE_IRON_SWORD	4

Heavier weights mark harder tiers (combat, iron-tier crafting, diamond collection), reflecting the long prerequisite chain in Crafter's tech-tree.

A.3.5 Crafter reward. Per step, the environment produces a scalar reward by combining (i) an achievement-unlock bonus weighted by the table above, (ii) a health delta, (iii) a small per-step survival bonus, and (iv) a death penalty. Let $a_{i,t} \in \{0, 1\}$ indicate whether the i-th achievement is unlocked at step t and $\Delta a_{i,t} = a_{i,t} - a_{i,t-1}$, let h_t be the player's health, and let $\mathbf{1}[\text{alive}_t]$, $\mathbf{1}[\text{died}_t]$ flag the alive/death transitions. The reward is

$$r_t \;=\; \sum_{i=0}^{21} w_i \cdot \Delta a_{i,t} \;+\; \kappa_h \cdot (h_t - h_{t-1}) \;+\; \rho_{\text{surv}} \cdot \mathbf{1}[\text{alive}_t] \;+\; \rho_{\text{death}} \cdot \mathbf{1}[\text{died}_t], \qquad (12)$$

with w_i taken from the weight column of the table in A.3.4, and $\kappa_h = 0.5$, $\rho_{\text{surv}} = 0.1$, $\rho_{\text{death}} = 0$ matching the default `EnvParams` used in all of our experiments. Each achievement contributes its weight at most once per episode, so the dominant component of the episodic return comes from the achievement bonus. The episodic *Game Reward* reported in Section 4.2 is the sum of r_t over a full episode; *Achievement Counts* is the number of distinct achievements with $a_{i,T} = 1$ at episode end.

A.3.6 Evaluation metrics. Four metrics are defined identically in all three games. *Reasoning Hallucination Rate* is $1 - Q(w_t, s_t)$ averaged over evaluated steps, where Q is the reasoning–state match rate of Section 3.4 computed against the same privileged attributes used for filtering. *Action Success Rate* is the fraction of executed actions the environment accepts, using the per-game criteria given above. *Game Reward* is the environment return accumulated over an episode. *Train Data* counts the supervised samples consumed by each method. *Survival Steps*, reported for bigfish and Crafter, is the number of steps before death or the step cap. The remaining entries of Table 1 are game-specific progress measures: winning rate for bigfish and MiniHack, and for Crafter the achievement count together with the tech-tree indicators *Best Material*, *Best Tool*, *Best Weapon* and *Best Defeated Creature*, which record how deep into the crafting hierarchy the agent progressed. All numbers are averaged over the evaluation episodes of the corresponding game.

A.4 Baselines and Implementation Details

All trainable methods share the Qwen3-VL-8B-Instruct backbone and differ only in their supervision signal and update rule. Every method is evaluated on the same 16 held-out seeds in ProcGen-bigfish and Crafter, and on 16 held-out seeds for each of the ten MiniHack tasks.

Zero-shot VLMs. Qwen3-VL-8B-Ins is the *un*-finetuned Qwen3-VL-8B-Instruct model, and Gemini-3.5-flash is queried through the Gemini API. In each environment both models use the same current-frame input, game-specific action specification and basic strategy prompt as STPI, but never receive privileged state; MiniHack additionally includes the same short textual action history used by all methods. Both models use greedy decoding (temperature $= 0$, top-$p = 1$), no in-context examples, and the failed-action feedback of Section 3.6.

ProcGen-bigfish. The policy uses the nine-action prompt of Appendix A.1, a maximum rollout length of 1,500 steps and learning rate 1×10^{-5}. SFT demonstrations are generated by a rule-based expert. Evaluation uses 16 held-out seeds.

MiniHack. The prompt exposes the task-specific 8–12 action set and corresponding task hint. Rollout limits are task-dependent (40–200 steps; Appendix A.3), and the learning rate is 1×10^{-6}. Rule-based experts generate SFT demonstrations separately for each task. We evaluate 16 episodes per task and macro-average over the ten tasks.

Crafter. The policy uses the 17-action prompt and strategy list of Appendix A.1, a maximum rollout length of 10,000 steps and learning rate 1×10^{-6}. SFT uses expert state–action pairs from a PPO-RNN teacher with superhuman performance, following Park et al. (2025). Evaluation uses 16 held-out seeds.

Table 4: Environment-specific settings. The learning rate applies to all trainable methods in the corresponding environment.

Setting	ProcGen-bigfish	MiniHack	Crafter
Learning rate	1×10^{-5}	1×10^{-6}	1×10^{-6}
Maximum rollout steps	1,500	40–200 (task-dependent)	10,000
Evaluation episodes	16	16 per task (160 total)	16
Action-set size	9	8–12 (task-dependent)	17
SFT data source	Rule-based expert	Rule-based experts	PPO-RNN expert

Shared training hyper-parameters. The remaining method-specific settings are shared across the three environments.

Table 5: Training hyper-parameters shared across environments. All methods use the Qwen3-VL-8B-Instruct backbone. "—" means the entry does not apply.

Hyper-parameter	SFT	GRPO	GiGPO	STPI (ours)
Fine-tuning scheme	Full	Full	Full	Full
Optimizer / precision	AdamW / bf16	AdamW / bf16	AdamW / bf16	AdamW / bf16
LR scheduler / warm-up	cosine / 0.1	constant	constant	cosine / 0.1
Per-device batch size	4	1 (micro)	1 (micro)	4
Gradient-accumulation steps	1	—	—	1
Mini-batch size	—	8	8	—
Rollout batch size	—	8	8	—
GPUs / nodes	8 / 1	8 / 1	8 / 1	8 / 1
Training epochs / iters	1 epoch	50 epochs	50 epochs	—
Max prompt / response length	4096 / —	4096 / 512	4096 / 512	4096 / —
Rollout group size n	—	8	8	—
Sampling temperature (rollout)	—	0.4	0.4	0.7
KL-loss coeff. β_{KL}	—	0.04	0.04	—
Invalid-action penalty	—	0.1	0.1	—
Reasoning filter τ	—	—	—	0.7
Replay base α	—	—	—	0.2
Replay decay β	—	—	—	0.8

SFT supervises only the action token from the environment-specific expert sources above. **GRPO** (Shao et al., 2024) uses episode-level game reward as the sole signal, with group-relative advantage estimation and a KL penalty to the reference policy. **GiGPO** (Feng et al., 2025) replaces the episode reward of GRPO with dense per-step game reward. **STPI** uses the conditional two-stage rollout and filtered self-distillation in Algorithm 1: at each iteration it mixes the current filtered samples with an experience-replay buffer sampled at rate $\rho_j^{(k)} = \alpha \cdot \beta^{k-j-1}$, then minimizes the NLL loss of Section 3.5. At inference time, all trained methods use the same game-specific base prompts and current-frame visual input; GRPO, GiGPO and STPI receive failed-action feedback, while SFT omits it because its demonstrations contain no environment outcomes.