GENERATIVE MODELS AS ONLINE POLICY CORRECTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

How can an agent learn efficiently online? Reward-based reinforcement learning can require costly exploration to discover better behaviors in open-world environments with sparse rewards. We propose an alternative in which an agent learns not only by reinforcing successful behavior but also by correcting mistakes in imagination online. We introduce ReVision, a learning algorithm that fully closes the loop in online corrective learning using pretrained generative models as online correctors. Given a policy rollout, ReVision identifies failures and generates counterfactual rollout continuations that correct them. The policy is then finetuned on these corrected rollouts and redeployed in the environment to collect new experience, repeating this cycle online. On Minecraft ObtainDiamond, ReVision substantially improves both behavior-cloned and vision-language-action policies. By applying pretrained priors at the learner's own failures, ReVision achieves returns comparable to PPO using $5\times$ fewer online environment steps (6M versus 30M). Controlled comparisons show that repeatedly correcting the updated learner yields greater improvements than distilling teacher demonstrations or correcting a frozen learner. Further experiments on manipulation tasks like RLBench also demonstrate improvements in pretrained manipulation policies. See visualizations of results at https://web.mk711winder.workers.dev/.

1 INTRODUCTION

Efficient online learning requires an agent to discover better behavior and incorporate it into a policy. Reinforcement learning addresses this through interaction and reward, using feedback on the outcomes of actions to guide policy improvement (Sutton & Barto, 1998). Pretrained behavioral priors can make learning more efficient (Baker et al., 2022), but exploration remains challenging when rewards are sparse and progress requires long sequences of consistent actions (Ecoffet et al., 2021). An agent may repeatedly encounter the same bottleneck without discovering how to overcome it. This motivates extracting more useful supervision from the attempts it has already made.

Learning by diagnosing and correcting failures offers a complementary route. Classical AI systems improved behavior by debugging unsuccessful procedures and explaining plan failures (Sussman, 1973; Hammond, 1986). Corrective imitation learning and DAgger similarly provide expert supervision on states visited by the learner (Chernova & Veloso, 2009; Ross et al., 2011). These approaches suggest that a failed attempt can be a starting point for constructing better behavior. However, they rely on explicitly specified debugging knowledge or access to an expert. The challenge is to generate informative corrections automatically for the failures encountered by an evolving policy.

Generative models offer a way to address this challenge. Vision-language models (VLMs) can reason about tasks and propose actions (Driess et al., 2023), while action-conditioned world models predict their consequences (Yang et al., 2024; Hafner et al., 2025b), enabling search over imagined futures (Du et al., 2024). Crucially, a world model can branch from a past failure and simulate alternative continuations without restoring the real environment to that state. This allows an agent, even in the physical world, to explore corrections that would otherwise require costly or infeasible retries. Building on policy learning from simulated experience (Sutton, 1991; Hafner et al., 2025b; Yang et al., 2026), we ask: *can generative models serve as online teachers that turn an agent's own failures into corrective training data?*

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.

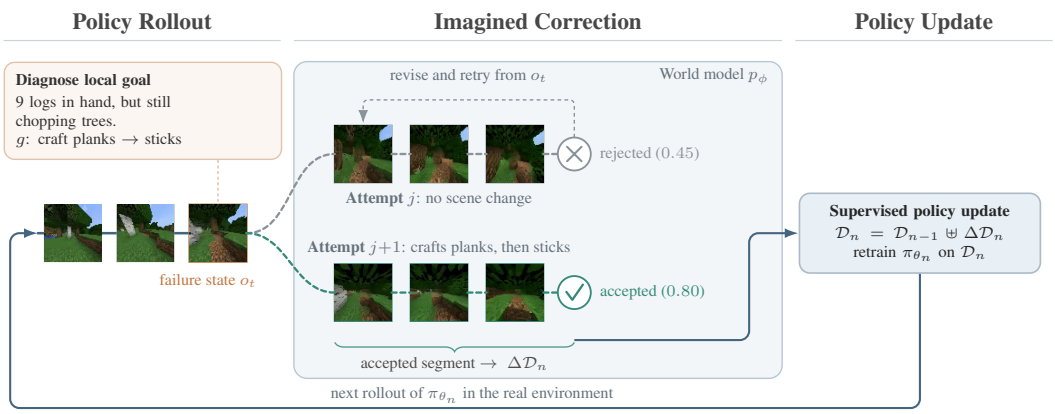


Figure 1: **ReVision method overview.** The learner policy rolls out in the environment, and a VLM diagnoses a failure and local goal from the current rollout. The corrector proposes alternative actions, the world model predicts their outcomes, and rejected attempts inform the next proposal. Continuations accepted through search update the policy, whose next rollout begins another round.

We introduce **ReVision**, an online policy improvement algorithm that learns by correcting mistakes in imagination. Given a rollout from the current policy, a VLM identifies failures and specifies local goals. The full loop is illustrated in figure 1. A corrector proposes alternative actions through a library of action primitives, while an action-conditioned world model simulates their consequences. VLM feedback guides revision, and continuations judged to advance the local goal are retained for training. The policy is fine-tuned on these corrections alongside its original demonstrations and redeployed, so subsequent corrections target the updated learner's failures. These local corrections allow generative models to provide useful supervision even when they cannot reliably solve the entire task.

We evaluate and demonstrate the efficacy of ReVision on Minecraft and robot manipulation tasks. Our contributions are: (i) ReVision, a learning algorithm that fully closes the loop in DAgger-style online corrective learning on a policy's own failures: generative models detect where the learner fails, propose corrections, and verify them in a world model, with no human or scripted expert in the loop; (ii) evidence that online correction for an evolving policy outperforms teacher distillation; and (iii) results on Minecraft ObtainDiamond and manipulation tasks, where ReVision substantially improves BC- and VLA-initialized policies, and on Minecraft reaches returns comparable to reward-based methods trained with five times as many environment steps.

2 Related Work

World Models for Policy Learning. PETS and PlaNet plan by evaluating candidate action sequences under learned dynamics, and MuZero and the EfficientZero series combine learned models with tree search (Chua et al., 2018; Hafner et al., 2019; Schrittwieser et al., 2020; Ye et al., 2021; Wang et al., 2024). Dyna, MBPO, and Dreamer learn from model-generated experience, and RISE improves robot policies using imagined rollouts weighted by a progress-value model (Sutton, 1991; Janner et al., 2019; Hafner et al., 2025a; Yang et al., 2026). In visual environments, UniSim and iVideoGPT learn interactive dynamics, DIAMOND and Dreamer4 train policies inside learned simulators, UniPi generates visual plans for control, and DreamZero and LingBot-VA jointly predict future observations and actions (Yang et al., 2024; Wu et al., 2024; Alonso et al., 2024; Hafner et al., 2025b; Du et al., 2023; Ye et al., 2026; Li et al., 2026b).

Foundation Models for Embodied Control. SayCan, Code as Policies, and PaLM-E use foundation models for grounded skill selection, control program generation, and embodied reasoning, respectively (Ahn et al., 2022; Liang et al., 2023; Driess et al., 2023), and HiP and VLP combine semantic planning with video prediction and goal-conditioned control (Ajay et al., 2023; Du et al., 2024). In Minecraft, VPT learns behavioral priors from video and STEVE-1 extends them to instruction following (Baker et al., 2022; Lifshitz et al., 2023), while MineDojo, Voyager, and MineDreamer provide guidance through video-language rewards, executable skill libraries, and imagined visual goals (Fan et al., 2022; Wang et al., 2023; Zhou et al., 2024). ReVision instead uses foundation models to construct corrective training data for an evolving policy.

Corrective Imitation and Online Policy Improvement. DAgger aggregates expert labels on learner-induced states, HG-DAgger enables human interventions, and DART injects noise during demonstrations to elicit recoveries (Ross et al., 2011; Kelly et al., 2019; Laskey et al., 2017). Recent methods use world models to support corrective supervision. WM-DAgger generates deviations and recoveries around expert trajectories (Yu et al., 2026). Dream2Fix synthesizes counterfactual failures with recovery labels derived from known perturbations, trains a VLM to predict corrections, and applies it to policy failures (Li et al., 2026a). Hi-WM collects human corrections at human-identified failures during imagined policy rollouts (Li et al., 2026c). ReVision couples correction generation to the evolving learner: an imperfect generative teacher diagnoses the current policy's failures and searches for local corrections, using rejected attempts to refine subsequent proposals. Accepted continuations supervise policy updates, changing the failures encountered in later rounds without human intervention in the online loop.

3 METHOD

ReVision improves a policy by turning its own failures into corrective training data. A policy trained on offline demonstrations can encounter states for which those demonstrations provide little guidance, causing distribution shift errors to compound over a rollout (Ross et al., 2011). ReVision aims to use each failure of the policy as the starting point to generate a short corrective continuation.

We use a vision-language model (VLM) to identify failures and specify local goals, a corrector to propose alternative actions, and a world model to predict their consequences. Accepted continuations are used to train the policy, whose subsequent rollouts determine the next correction problems. This turns planning in imagination into an online source of supervision targeted to the learner's evolving behavior. Figure 1 illustrates the overall procedure. After introducing the policy and its initialization (Sec. 3.1), we describe how failures define local correction goals (Sec. 3.2), how we search for corrective continuations (Sec. 3.3), and how these continuations update the policy (Sec. 3.4).

3.1 PROBLEM SETUP AND POLICY INITIALIZATION

We consider a partially observed environment with observations o_t, low-level actions $a_t \in \mathcal{A}$, and dynamics $p^\star(o_{t+1} \mid o_{\le t}, a_{\le t})$. The policy $\pi_\theta(a_t \mid o_{\le t})$ acts on its available observation context, including visual inputs and any task-state features supplied to it. We initialize policy π_{θ_0} by supervised training on offline demonstrations $\mathcal{D}_0$. A horizon-T rollout is $\tau = (o_0, a_0, \ldots, o_T)$, and $c_t = (o_0, a_0, \ldots, o_t)$ denotes the prefix ending before action a_t. We let x_t denote any additional structured task state at time t that is carried alongside the visual context.

3.2 IDENTIFYING POLICY FAILURES

ReVision begins by performing rollouts to identify failure states of the current model. At round n, we collect a batch $\mathcal{B}_n$ of horizon-T rollouts from $\pi_{\theta_{n-1}}$. We divide each rollout into contiguous chunks, giving the diagnostic model temporal context for recognizing stalled progress or repeated ineffective behavior. Furthermore, a mistake in a long-horizon task is rarely one action and is better presented as short-horizon behavior. A scoring VLM F_ω assigns each chunk a correction priority. We select the highest-priority chunks for correction, and for each selected chunk beginning at t, the model explains the failure and specifies a local goal g_t.

The local goal converts a failure into a concrete teaching problem. For example, if an agent places a crafting table but resumes gathering wood despite having the required materials, a useful goal is to craft a wooden pickaxe. A continuation achieving this goal can teach the missing behavior without also demonstrating how to obtain diamonds. We hold the prefix c_t fixed and perform search from its endpoint for a correction that advances g_t. Hence, we focus our supervision on improving a specific identified failure state with a clear goal.

3.3 SEARCHING FOR CORRECTIONS

For the correction generation process for a given failure state, we define a search procedure that returns either an accepted continuation or no correction. For each selected failure, we use the same sequence of proposal, execution, and evaluation. The search returns the first accepted continuation or no correction if its attempt budget is exhausted.

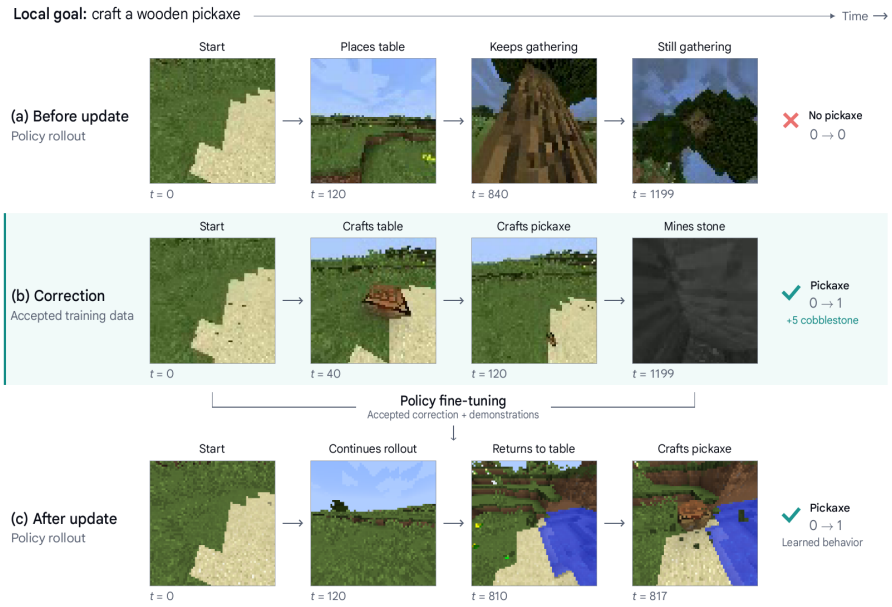


Figure 2: **Visual ReVision training sample**. From the same task state, the (a) original policy places a crafting table but returns to gathering and never crafts a pickaxe. (b) The correction instead crafts a wooden pickaxe and mines five cobblestone. The policy is fine-tuned on the correction in the replay buffer. (c) In a subsequent rollout, the updated policy performs the pickaxe-crafting behavior.

Proposing actions. A VLM corrector C_ψ receives the task description, prefix c_t, auxiliary state x_t, local goal g_t, and a history h of rejected attempts. It proposes a short sequence of action primitives to achieve the goal. The corrector can then be queried again using the resulting observations and state to extend the candidate trajectory, until the correction is complete or the correction horizon H is reached. The Minecraft library includes movement, camera control, mining, and visual aiming, as well as crafting and smelting macros. The action primitive interface lets the corrector propose behavior at the level of reusable skills, while still retaining the low-level actions needed to interface with the environment. Appendix C (Table 17) provides more details on the primitives.

Generation. To view the outcome of the proposed actions, we initialize the correction backend at the selected branch point and generate the sequence. With $p_{\text{corr}} = p_\phi$, the world model conditions on the prefix, auxiliary state, and executed actions to predict subsequent observations and state features. The corrector can use these outcomes to propose additional action primitive sequences without environment interaction. We denote this process by $\text{Generate}(C_\psi, p_{\text{corr}}, c_t, x_t, g_t, h, H)$ in Algorithm 1. The resulting trajectory $\widetilde{\tau}$ contains the action-produced observations and states.

Evaluation and search. After action proposal, there is no guarantee that directly applying corrector C_ψ is sufficient to achieve g_t. Hence, we require some form of search. A VLM scorer examines the candidate trajectory's observations and state changes for progress toward g_t. We denote the resulting acceptance decision by $S_\nu(c_t, x_t, g_t, \widetilde{\tau}) \in \{0, 1\}$. If accepted, the observation–action trajectory of the correction is added to the replay buffer. Otherwise, if the attempt is rejected, its actions and outcomes are added to the corrector's context, and the next attempt restarts from the same branch point. Every attempt therefore revises the behavior after the same prefix, and rejected outcomes inform the next proposal without entering policy training. We allow at most J attempts per failure and retain the first accepted candidate. If all attempts are rejected, the branch contributes no supervision.

World-model instantiation. The world model p_ϕ predicts how the environment evolves under the proposed actions. To construct a continuation from time t, we condition on the fixed rollout prefix $c_t = (o_0, a_0, \ldots, o_t)$ and generate the subsequent observations and states. For Minecraft, p_ϕ is a multimodal diffusion transformer (mmDiT) (Esser et al., 2024) over frozen-VAE image latents (Rombach et al., 2022) and a 27-dimensional state vector. The image stream uses Diffusion Forcing (Chen et al., 2024), and the state stream predicts per-frame state changes, with joint objective

$$\mathcal{L}_{\text{WM}}(\phi) = \mathcal{L}_{\text{DF}}(\phi) + \lambda_s \mathcal{L}_{\text{state}}(\phi) \tag{1}$$

Algorithm 1 Online policy improvement with ReVision.

Require: Initial policy π_{θ_0}, demonstrations $\mathcal{D}_0$
Require: Failure scorer F_ω, corrector C_ψ, scorer S_ν, correction backend p_{corr}
Require: Training rounds N, correction attempts J, correction horizon H

$\mathcal{D}_0^{\text{corr}} \leftarrow \varnothing$
for $n = 1, \ldots, N$ **do**
  Collect rollouts $\mathcal{B}_n$ using $\pi_{\theta_{n-1}}$
  $\mathcal{F}_n \leftarrow \text{SelectFailures}(\mathcal{B}_n)$
  for each failure context and goal $(c_t, x_t, g_t) \in \mathcal{F}_n$ **do**
    $h \leftarrow \varnothing$
    for $j = 1, \ldots, J$ **do**
      $\widetilde{\tau} \leftarrow \text{Generate}(C_\psi, p_{\text{corr}}, c_t, x_t, g_t, h, H)$
      if $S_\nu(c_t, x_t, g_t, \widetilde{\tau}) = 1$ **then**
        Add $\widetilde{\tau}$ to $\mathcal{D}_n^{\text{corr}}$
        break
      end if
      Append the rejected attempt $\widetilde{\tau}$ to h
    end for
  end for
  Update π_{θ_n} on $\mathcal{D}_0$ and $\mathcal{D}_n^{\text{corr}}$ using Eq. 2
end for
return π_{θ_N}

Both streams condition on context and actions. The main experiments use WM-B; Table 4 and Appendix A.1 provide the configurations, training setup, and sampling procedure. For world models in the manipulation environments, we use a state prediction world model and an environment processor that converts predicted states into RGB frames for the policy.

3.4 Online Policy Improvement

We can use the ReVision corrections over multiple rounds to iteratively find failure states and improve the policy. At round n, we collect a batch of horizon-T rollouts from $\pi_{\theta_{n-1}}$, diagnose their chunks, and select the highest-priority failures for correction. Let $\Delta\mathcal{D}_n$ contain the observation–action pairs from all accepted continuations in that round. We accumulate these pairs in a correction buffer given by $\mathcal{D}_n^{\text{corr}} = \mathcal{D}_{n-1}^{\text{corr}} \uplus \Delta\mathcal{D}_n$, $\mathcal{D}_0^{\text{corr}} = \varnothing$.

Learning from corrections. The policy learns the low-level actions through behavior cloning. For a supervised trajectory dataset $\mathcal{D}$, the objective is $\mathcal{L}_{\text{BC}}(\theta; \mathcal{D}) = -\mathbb{E}_{(o_{\le t}, a_t)\sim\mathcal{D}}\left[\log \pi_\theta(a_t \mid o_{\le t})\right]$. We sample the correction buffer using a replay distribution q_n^{corr} that down-weights older corrections as the policy changes, with per-round mixture weights proportional to λ^{n-i} with round i and decay factor $\lambda \in (0, 1]$. The update gives equal weight to the original data and corrections:

$$\mathcal{L}_n(\theta) = \frac{1}{2}\mathcal{L}_{\text{BC}}(\theta; \mathcal{D}_0) + \frac{1}{2}\mathbb{E}_{(o_{\le t}, a_t)\sim q_n^{\text{corr}}}\left[-\log \pi_\theta(a_t \mid o_{\le t})\right] \tag{2}$$

The demonstration term mitigates forgetting of the policy's existing skills, while the correction term fits new behavior at its observed failures. If the correction buffer is empty, we train only on $\mathcal{D}_0$. After updating the policy, we collect new rollouts and repeat: each round can expose different failures as the policy reaches new states. An optional meta-scorer summarizes previous rounds and adjusts diagnostic priorities to focus subsequent correction effort; examples appear in Appendix B.6.

Execution variants. The same loop supports three variants. The primary variant, ReVision, collects policy rollouts in the environment and generates corrections in the world model. The fully simulated variant, ReVision-W, uses the world model for both stages, requiring no additional training-time environment interaction after offline initialization. The environment-executed reference, ReVision-E, uses the environment for both stages and restores checkpoints to generate alternative continuations from a selected failure. Hence, ReVision-E assumes the environment can be reset to any visited state, which is available in simulators but not in the real world. ReVision, the deployable variant, requires only forward rollouts: all branching happens in the world model.

4 EXPERIMENTS

Our experiments address three questions: whether corrective post-training improves pretrained policies across tasks (Sec. 4.1), what makes online correction effective (Sec. 4.2), and how replacing environment-executed corrections with imagined corrections affects learning (Sec. 4.3).

Experimental setup. Our primary benchmark is Minecraft ObtainDiamond in MineRL 0.4.4 (Guss et al., 2019). We report mean and median episode return under the milestone reward used by VPT (Baker et al., 2022), together with tool-tier success: the average success rate across cobblestone, stone pickaxe, furnace, iron ore, iron ingot, and iron pickaxe. These metrics measure intermediate task progress. Our policies are trained in a lower-data regime: MineRL v0.4.4 provides 4.1M demonstration frames for this task, roughly 45× fewer than the 2.5K hours of VPT contractor data used by Dreamer 4 (Hafner et al., 2025b), and far less than VPT's web-scale pretraining.

We initialize a ResNet-based BC policy and a Gemma 3-based VLA policy from 4.1M demonstration frames; both receive images and inventory state. Learned-model experiments use WM-B, a 406.6M-parameter mmDiT pretrained on 46.5M VPT frames and fine-tuned on 4.1M demonstration frames plus 4M synthetic frames. World-model training details appear in Appendix A.1. RLBench is our primary benchmark for robotics manipulation tasks. In this work, we evaluate five RLBench tasks (Insert Peg, Place Cups, Sort Shape, Stack Blocks, and Stack Cups). Manipulation settings are described alongside their results below.

4.1 POLICY IMPROVEMENT ACROSS TASKS

Minecraft policy improvement. Table 1 compares the three variants with PPO and Dreamer-style REINFORCE under BC and VLA initializations. The main comparison uses 6M rollout steps, with environment and world-model budgets matched in steps (1 world-model step is 3 environment ticks). PPO provides a reference for reward-based environment interaction, while Dreamer-style REINFORCE learns inside the same pretrained world model. Baseline implementations and training configurations are provided in Appendix B.1 and Appendix B.2.

	BC initialization			VLA initialization		
Method	Mean ± SEM	Median	Tool-tier success (%)	Mean ± SEM	Median	Tool-tier success (%)
Base policy	36.5 ± 4.5	19.0	14.5 ± 2.0	73.5 ± 12.2	35.0	28.3 ± 3.2
Post-training for 6M steps						
ReVision-E	104.4 ± 11.2	67.0	37.3 ± 2.9	114.8 ± 13.8	99.0	36.7 ± 3.2
ReVision	102.3 ± 11.1	67.0	36.3 ± 3.1	110.5 ± 13.5	67.0	36.5 ± 2.8
ReVision-W	85.8 ± 10.3	67.0	31.3 ± 2.6	94.5 ± 11.5	67.0	32.2 ± 2.8
PPO	77.8 ± 10.5	51.0	28.0 ± 2.4	78.9 ± 13.7	35.0	30.8 ± 3.1
Dreamer REINFORCE	77.5 ± 11.4	35.0	26.2 ± 2.8	78.3 ± 11.1	35.0	26.3 ± 2.9
Reference post-training for 30M steps:						
PPO (30M)	103.6 ± 11.9	67.0	37.0 ± 2.9	106.3 ± 12.9	67.0	36.4 ± 2.6
Reference without post-training:						
Open-loop VLM (GPT 5.6): mean 88.6 ± 11.4, median 67.0						

Table 1: **Minecraft performance.** BC and VLA base policies are shown before post-training, and the remaining initialized policies receive 6M matched post-training steps. Evaluations use 100 ObtainDiamond episodes across three seeds. ReVision-W and Dreamer-style RL use matched steps in the world-model, the remaining post-training methods use real-environment steps. Mean returns are reported with SEM computed over episodes. Tool-tier success is reported in percent.

ReVision increases mean return from 36.45 to 102.32 under BC initialization and from 73.50 to 110.50 under VLA initialization. Both initialized policies achieve higher mean returns than the reported reward-based references at this budget. ReVision-W also improves both policies, showing that corrective post-training can be effective using entirely imagined training rollouts. We analyze differences between the three execution variants in Sec. 4.3. The table also includes an open-loop

Method	Insert Peg	Place Cups	Sort Shape	Stack Blocks	Stack Cups	Avg.
PerAct (Shridhar et al., 2022)	5.6±4.1	2.4±3.2	16.8±4.7	26.4±3.2	2.4±2.0	10.7
Act3D† (Gervet et al., 2023)	27.0	3.0	8.0	12.0	9.0	11.8
3D Diffuser Actor (Ke et al., 2024)	65.6±4.1	24.0±7.6	44.0±4.4	68.3±3.3	47.2±8.5	49.8
SAM2Act (Fang et al., 2025)	84.0±5.7	47.0±6.0	**64.0**±4.6	76.0±8.6	78.0±4.0	69.8
BridgeVLA (Li et al., 2025)	**91.2**±1.8	**58.4**±4.6	55.2±5.2	84.8±9.1	**88.8**±3.3	**75.7**
RVT-2 (base) (Goyal et al., 2024)	40.0±0.0	38.0±4.5	35.0±7.1	80.0±2.8	69.0±5.9	52.4
RVT-2 + ReVision	43.2±3.0	46.4±7.0	44.8±8.2	**86.4**±2.0	78.4±2.0	59.7
RVT (base) (Goyal et al., 2023)	11.2±3.0	4.0±2.5	36.0±2.5	28.8±3.9	26.4±8.2	21.3
RACER‡ (Dai et al., 2025)	9.6±4.8	6.4±4.1	25.6±4.1	15.2±3.0	41.6±5.4	19.7
RVT + ReVision	20.0±5.7	15.2±3.9	32.0±3.6	36.8±3.9	34.4±5.4	27.8

Table 2: **RLBench results.** Multi-task success rate (%) on five difficult RLBench (James et al., 2020) tasks, with mean ± std over evaluation seeds. Avg. is the unweighted mean over the five tasks, computed by us. The top block lists prior methods as reported in Li et al. (2025) (†Act3D reports no std). RVT and RVT-2 base results are taken from Goyal et al. (2023) and Goyal et al. (2024), and RACER results from Dai et al. (2025) (‡modified RVT actor guided by a VLM supervisor at test time). The lower blocks group each ReVision result with its starting policy; ReVision rows are evaluated by the same protocol.

VLM baseline using the same observation inputs and action primitives. It executes action primitive proposals directly by receiving the same input as policies, without the correction-search procedure. The updated BC and VLA policies attain higher mean returns than this baseline at matched evaluation episode counts and lengths. This comparison concerns direct VLM execution; the effect of distilling teacher-generated data is examined separately in Sec. 4.2.

Online interaction efficiency. Table 1 also compares corrective post-training at 6M environment steps with PPO at 6M and 30M steps. ReVision reaches mean returns comparable to PPO trained with five times as many environment steps under BC initialization, and a higher mean return under VLA initialization. ReVision-E shows a similar pattern. These results demonstrate an interaction advantage under the reported budgets. The correction pipeline additionally uses a pretrained VLM, an action primitive library, and correction-generation compute; VLM usage is reported in Appendix B.3.

Robotic manipulation. We evaluate the correction procedure on five RLBench tasks: Insert Peg, Place Cups, Sort Shape, Stack Blocks, and Stack Cups. We focus on these tasks because they remain unsaturated for current policies, and are consistently among the lowest-success tasks for prior methods. We initialize the policies from RVT-1 and RVT-2 for ReVision with four RGB-D camera views. We also compare with RACER (Dai et al., 2025), the closest offline prior approach, which generates failures by perturbing expert actions and using a VLM to issue language corrections to the policy. The updated RVT-2 achieves 59.7% average success across these tasks, compared with the reported RVT-2 reference of 52.4%, and the updated RVT-1 similarly outperforms RVT-1 and RACER. The other methods in the table provide context from published multi-task RLBench evaluations. We also evaluate four MetaWorld v3 tasks (Yu et al., 2020), and results can be found in Appendix B.5.

4.2 Understanding Online Corrective Learning

We next examine whether improvement comes from additional optimization, teacher-generated data, or corrections targeted to the evolving learner. These controls use ReVision-E to remove learned-dynamics errors as a confound, and ensure differences between arms reflect the correction procedure alone. All arms share the corrector and action primitives and are evaluated at matched budgets of 250k and 500k additional policy updates.

Correcting an evolving learner. Table 3 compares the full online procedure with three controls: continued BC on the original demonstrations, BC on VLM demonstrations generated from random starts, and BC on corrections of a frozen learner's rollouts. The frozen-learner control uses the same correction procedure and matches generated correction volume while keeping the rollout policy fixed at its initial checkpoint. At 500k updates, the respective mean returns are 32.8, 61.0, and 65.2, compared with 91.6 for full ReVision-E. The strongest result comes from repeatedly correcting the updated learner, supporting the value of online corrective supervision as the policy changes.

Training setting	250k policy updates		500k policy updates	
	Mean ± SEM	Median	Mean ± SEM	Median
Offline BC				
Original demonstrations	28.3 ± 3.8	19.0	32.8 ± 8.3	11.0
VLM demonstrations from random starts	34.8 ± 4.9	35.0	61.0 ± 9.8	35.0
VLM corrections from policy rollouts	69.1 ± 13.8	67.0	65.2 ± 8.7	67.0
Online ReVision-E				
Without goal setting	73.5 ± 10.1	67.0	54.0 ± 7.5	51.0
Without acceptance gate	67.1 ± 14.7	51.0	60.5 ± 10.5	35.0
Full ReVision	82.3 ± 15.0	67.0	91.6 ± 11.9	67.0
Base policy (no additional training): mean 36.5 ± 4.5; median 19.0				

Table 3: **Ablations.** Results are reported after 250k and 500k additional policy updates. Offline BC uses the original demonstrations, VLM demonstrations generated from random starts, or VLM corrections of frozen policy rollouts. Online revision compares ReVision with variants that remove goal setting or the acceptance gate. Evaluations use 100 episodes per evaluation. We report mean return ± SEM over episodes and median return.

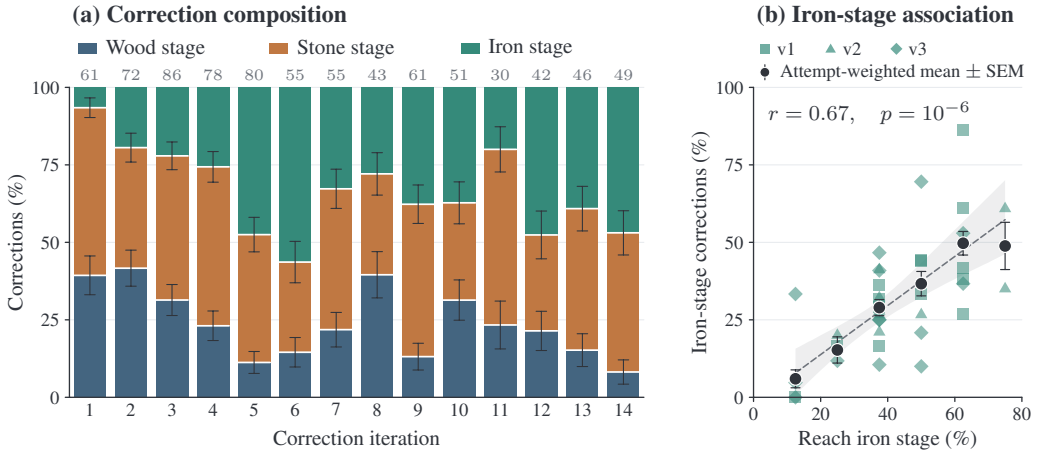


Figure 3: **Correction distribution across online training.** Runs performed on VLA policy across three training seeds. **(a)** Each bar shows the distribution of correction attempts across the task stage progression, computed from the state at the end of the chunk. Numbers above the bars represent attempt counts. Error bars are ±1 SEM of the share. **(b)** Iron-stage share of each iteration's corrections versus the iteration's iron-stage reach rate. Dark points represent attempt-weighted means at each reach level (±1 SEM). The correlation holds within runs ($r = 0.66$), meaning corrections shift toward later stages as online training progresses.

Figure 3 illustrates this changing supervision distribution. As training progresses and the learner reaches later stages, correction attempts increasingly address later-stage behavior. This shows how correction targets evolve; the frozen-learner comparison tests the benefit of updating those targets.

Local goals and correction selection. We also remove two components of online ReVision-E. The no-goal variant omits the local goal g_t from the corrector input. The no-acceptance variant retains every first proposal, jointly removing rejection and subsequent retries. At 250k updates, the full method has higher mean return, although the differences are modest relative to the reported uncertainty. By 500k updates, full ReVision-E improves to 91.6, while the no-goal and no-acceptance variants decline to 54.0 and 60.5, respectively. This pattern supports the combined role of goal-directed proposals and correction selection in sustaining improvement across rounds.

4.3 Learning from Imagined Corrections

Section 4.2 used environment dynamics to isolate the correction procedure. However, ReVision-E requires restoring the environment to arbitrary visited states, which physical systems generally do not. The world model in ReVision removes this requirement: it provides the branching in simulation, so corrections can be imagined from failure states the real environment cannot return to. Figure 4 compares execution variants and system configurations.

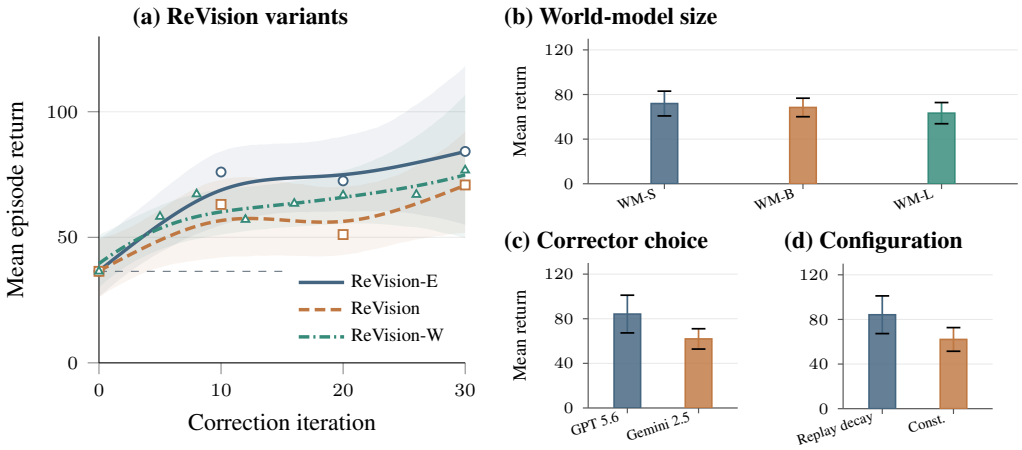


Figure 4: **ReVision variants and mechanism ablations.** (a) Training curves for the environment–environment (ReVision-E), environment–world-model (ReVision), and world-model–world-model (ReVision-W) configurations. Markers show checkpoint means and shaded regions show 95% pointwise bootstrap CI. (b–d) Measured returns at correction iteration 30 of independent runs, with error bars showing $\pm$ one SE over 50 evaluation episodes and three training seeds. (b) Compares world-model sizes for ReVision, (c) compares correctors for ReVision-E with other components held fixed, and (d) compares performance with the replay decay mechanism.

Replacing environment corrections with imagination. Figure 4(a) compares ReVision-E, ReVision, and ReVision-W across correction iterations. ReVision-E generally achieves higher mean returns, although uncertainty overlaps across variants. ReVision retains substantial policy improvement when corrections are generated by the world model, while ReVision-W also improves without additional training-time environment interaction. Together with the main results, these comparisons show that imagined corrections can provide useful supervision, while leaving a performance gap relative to environment-executed corrections.

World-model fidelity and scale. Appendix A.2 compares world-model predictions with environment outcomes on scripted crafting, mining, and composite interactions. Agreement varies across tasks, with both optimistic and pessimistic errors. These measurements characterize model limitations, while downstream policy evaluations assess whether the generated supervision is useful for learning. Figure 4(b) compares WM-S, WM-B, and WM-L under BC ReVision. Larger models with better metrics do not necessarily yield a clear improvement in downstream return under the evaluated setup.

Corrector and replay configuration. Figure 4(c) compares corrector models under ReVision-E specified in Table 13, changing the VLMs used for diagnosis, action proposals, and scoring together. Figure 4(d) contrasts age-decayed correction sampling with a constant-weight configuration, and shows improvement with replay decay. These results assess sensitivity to the teacher and replay design alongside the main comparisons of online supervision and correction backends.

5 Discussion

We introduced ReVision, an online policy-improvement method that turns a policy's failures into training data. On Minecraft ObtainDiamond and robotic tasks, ReVision improves both BC- and VLA-initialized policies. Relative to reward-based post-training, ReVision exceeds PPO and Dreamer-style RL at matched environment budgets and matches or exceeds PPO trained with five times as many environment steps, by drawing on pretrained generative models that reward-based methods do not use. Controlled comparisons that share the teacher and primitives show that the largest gain comes from our online correction method, beyond distilling the teacher or correcting a fixed policy.

Limitations. ReVision is limited by what the VLM can diagnose and what the corrector can repair within a short horizon. Hence, failures that require sustained behavior, such as surviving while mining, may not be recoverable from a single correction. A hierarchical variant, where the VLM proposes multiple intermediate subgoals for the corrector, could extend corrections to longer horizons. Corrections are also evaluated in a world model whose simulation contains some bias. This bias could be reduced by periodically fine-tuning the world model on the agent's own real rollouts, or by using ensemble disagreement to reject corrections in regions the model predicts poorly.

Reproducibility Statement

Section 3 specifies the ReVision training loop, including (i) how rollouts are chunked and scored, (ii) how correction problems are constructed at failure states, (iii) the retry-and-accept mechanism, and (iv) how accepted corrections add to the replay buffer and used for policy updates.

Appendix A.1 reports the world-model architecture, tokenization, training corpus sizes, and optimization settings, with model-scale summaries in Table 4 and shared architecture/sampling settings in Table 5. Appendix B.1 contains details on baseline training and implementation. Appendices B.2 and B.3 list key ReVision hyperparameters (Table 10), the VLM backends used by each corrector arm (Table 13), aggregate VLM usage statistics (Table 14), and compute accounting. For robot manipulation, and B.4 detail protocol and results. Appendix C specifies all action primitives and how corrections are mapped into each student's native action space.

AI Use Statement

Generative AI tools assisted with manuscript organization, language editing, literature lookup, and figure generation. AI tooling also assisted with experiment setup, programming, and analysis. All AI-assisted outputs were reviewed and verified by the authors, who retain full responsibility for the accuracy, interpretation, and integrity of the manuscript and its results.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*, 2022. doi:10.48550/arXiv.2204.01691. 2

Anurag Ajay, Seungwook Han, Yilun Du, Shuang Li, Abhi Gupta, Tommi Jaakkola, Josh Tenenbaum, Leslie P. Kaelbling, Akash Srivastava, and Pulkit Agrawal. Compositional foundation models for hierarchical planning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. 2

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in Atari. In *Advances in Neural Information Processing Systems*, volume 37, 2024. 2

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 (NeurIPS)*, 2022. 1, 2, 6, 17

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 4

Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009. 1

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31, 2018. 2

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. RACER: Rich language-guided failure recovery policies for imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 7

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023. 1, 2

Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. 2

Yilun Du, Sherry Yang, Pete Florence, Fei Xia, Ayzaan Wahid, Brian Ichter, Pierre Sermanet, Tianhe Yu, Pieter Abbeel, Joshua B. Tenenbaum, Leslie Kaelbling, Andy Zeng, and Jonathan Tompson. Video language planning. In *International Conference on Learning Representations (ICLR)*, pp. 31138–31155, 2024. 1, 2

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021. 1

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 4

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems*, volume 35, 2022. 2

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating visual foundation model with a memory architecture for robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2025. 7

Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3D: 3d feature field transformers for multi-task robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2023. 7

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. RVT: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning (CoRL)*, 2023. 7

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. RVT-2: Learning precise manipulation from few demonstrations. In *Robotics: Science and Systems (RSS)*, 2024. 7

William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of minecraft demonstrations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019. 6

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 2019. 2

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025a. doi:10.1038/s41586-025-08744-2. 2, 17

Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025b. doi:10.48550/arXiv.2509.24527. 1, 2, 6

Kristian J. Hammond. CHEF: A model of case-based planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 267–271, 1986. 1

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL https://jmlr.org/papers/v23/21-1342.html. 17

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. 7

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 2

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3D Diffuser Actor: Policy diffusion with 3d scene representations. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, 2024. 7

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019. doi:10.1109/ICRA.2019.8793698. 3

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. DART: Noise injection for robust imitation learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 143–156. PMLR, 2017. URL https://proceedings.mlr.press/v78/laskey17a.html. 3

Dayou Li, Jiuzhou Lei, Hao Wang, Lulin Liu, Yunhao Yang, Zihan Wang, Bangya Liu, Minghui Zheng, and Zhiwen Fan. Learning actionable manipulation recovery via counterfactual failure synthesis. *arXiv preprint arXiv:2603.13528*, 2026a. doi:10.48550/arXiv.2603.13528. 3

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026b. doi:10.48550/arXiv.2601.21998. 2

Peiyan Li, Yixiang Chen, Hongtao Wu, Xiao Ma, Xiangnan Wu, Yan Huang, Liang Wang, Tao Kong, and Tieniu Tan. BridgeVLA: Input-output alignment for efficient 3d manipulation learning with vision-language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 7

Yaxuan Li, Zhongyi Zhou, Yefei Chen, Yanjiang Guo, Jiaming Liu, Shanghang Zhang, Jianyu Chen, and Yichen Zhu. Hi-WM: Human-in-the-world-model for scalable robot post-training, 2026c. URL https://arxiv.org/abs/2604.21741. 3

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. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023. 2

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. STEVE-1: A generative model for text-to-behavior in Minecraft. *arXiv preprint arXiv:2306.00937*, 2023. doi:10.48550/arXiv.2306.00937. 2

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022. 4

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. 1, 3

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 (7839):604–609, 2020. 2

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347. 17

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2022. 7

Gerald J. Sussman. *A Computational Model of Skill Acquisition*. PhD thesis, Massachusetts Institute of Technology, 1973. 1

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991. doi:10.1145/122344.122377. 1, 2

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998. 1

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 2

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. EfficientZero V2: Mastering discrete and continuous control with limited data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51041–51062. PMLR, 2024. 2

Jialong Wu, Shaofeng Yin, Ningya Feng, Xu He, Dong Li, Jianye Hao, and Mingsheng Long. iVideoGPT: Interactive VideoGPTs are scalable world models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. 2

Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, et al. RISE: Self-improving robot policy with compositional world model. *arXiv preprint arXiv:2602.11075*, 2026. 1, 2

Sherry Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. In *International Conference on Learning Representations (ICLR)*, 2024. 1, 2

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026. doi:10.48550/arXiv.2602.15922. 2

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering Atari games with limited data. In *Advances in Neural Information Processing Systems*, volume 34, 2021. 2

Anlan Yu, Zaishu Chen, Peili Song, Zhiqing Hong, Haotian Wang, Desheng Zhang, Tian He, Yi Ding, and Daqing Zhang. WM-DAgger: Enabling efficient data aggregation for imitation learning with world models. *arXiv preprint arXiv:2604.11351*, 2026. doi:10.48550/arXiv.2604.11351. 3

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2020. 7, 20

Enshen Zhou, Yiran Qin, Zhenfei Yin, Yuzhou Huang, Ruimao Zhang, Lu Sheng, Yu Qiao, and Jing Shao. MineDreamer: Learning to follow instructions via chain-of-imagination for simulated-world control. *arXiv preprint arXiv:2403.12037*, 2024. doi:10.48550/arXiv.2403.12037. 2

A Additional Experimental Details and Results

A.1 World Model Architecture and Training

The Minecraft world model p_ϕ is a multimodal diffusion transformer (mmDiT) operating on VAE latents and raw state vectors. A frozen convolutional VAE encodes each 64×64 RGB observation to an $8 \times 8 \times 16$ latent. The transformer uses patch size two, producing a 4×4 grid of 16 visual tokens per frame. We define the world-model time step to be a fixed frame stride of 3 game ticks (action_stride $= 3$): each world-model step conditions on a stack of the 3 raw actions taken between the two kept frames. With 26 context frames at this stride, a training sequence contains 416 visual tokens and 26 state slots. Positions use 3D RoPE.

Model	Width	Layers	Heads	Parameters
WM-S	768	10	12	160.6M
WM-B	1280	10	20	406.6M
WM-L	1280	18	20	708.7M

Table 4: **Minecraft world-model configs.** Parameter counts include the trainable mmDiT and frozen VAE.

Component	**Setting**
Observation resolution	64×64 RGB
VAE	convolutional VAE, base channels 64, multipliers $[1, 2, 2, 4]$, three residual blocks
VAE latent	$8 \times 8 \times 16$ ($8\times$ spatial downsampling), frozen
Patch size / tokens per frame	2 / 16
Context / rollout horizon	26 frames total, stride 3 ticks (action_stride $= 3$; stack 3 actions/step)
Sequence length	416 visual $+26$ state $= 442$ tokens
Positional encoding	3D RoPE; no learned positional table
Attention head dimension	64
MLP ratio	4.0
Action embedding	$d_{\text{action}} = 128$; camera 41 bins, maximum 25°
Raw state	$d_{\text{state}} = 27$; MoT width $0.125d$, MLP ratio 4.0
Diffusion	1000 training steps, 50 sampling steps, σ-preconditioning 0.125

Table 5: **World model architecture and sampling settings.** The table lists observation and latent representations, transformer settings, and diffusion parameters.

All world models are trained in two stages with the VAE frozen. Stage 1 pretrains on VPT gameplay with the state loss disabled. Stage 2 fine-tunes on the task-specific MineRL corpus with the state loss enabled and a lower learning rate. 4M frames of stage 2 training data was generated offline through VLM rollout on partly scripted sequences in the environment for world model training. These frames are only used in world model training to model additional interactions, and are not used in policy training. Inference uses EMA weights.

Conditioned corrective rollout. Prefix frames are held fixed while future frames are denoised. With K reverse-diffusion steps, the schedule at sampling step $m \in \{0, \dots, K\}$ and trajectory index τ is

$$\mathcal{K}_{m,\tau} = \begin{cases} 0, & \tau \le t \quad \text{(fixed context)}, \\ K - m, & \tau > t \quad \text{(generated continuation)}. \end{cases} \tag{3}$$

In ReVision-W, the conditioning prefix is itself model-generated. Future frames are initialized from noise and generated conditioned on the prefix and proposed actions. Longer candidates can be generated in non-overlapping blocks conditioned on the preceding block.

A.2 World Model Dynamics Analysis

We examine the world model's ability to model environment interactions by running the world model simulations against the environment simulation. This is directly related to the ability to simulate dynamically accurate correction trajectories for each necessary interaction. Running 320 episodes across 16 tasks, we evaluate correction agreement between the world model and environment. In each episode, the environment is reset with a fixed seed and a task-specific starting inventory, and the

	Stage 1: pretraining	**Stage 2: fine-tuning**
Data	VPT gameplay	task-specific MineRL corpus
Corpus size	46.5M frames	8.1M frames
Trajectories	10,414	1,117
Training windows	5,659,248	1,224,845
Optimizer steps	450,000	459,324
Epochs over windows	3.8	18.0
Initialization	random	stage-1 weights
Learning rate	2×10^{-4}	5×10^{-5}
Warmup steps	6,000	1,000
State loss weight	0.0 (disabled)	1.0
Shared: AdamW, $\beta = (0.9, 0.99)$, weight decay 10^{-4}; batch size 48; gradient clipping 1.0 linear warmup then constant LR; mixed precision; EMA decay 0.9999; min-SNR $\gamma = 5$ context dropout 0.1 visual / 0.1 state; action/state dropout 0.01/0.1; state-loss warmup 10,000 steps		

Table 6: **World model training configuration.** Stage-specific settings are shown for pretraining and fine-tuning, followed by shared optimization settings.

world model is initialized from the same first observation. The same scripted action sequence is then executed open-loop in both. The 16 tasks are defined as

- **Craft (8 tasks):** Single crafting or smelting operations given sufficient ingredients: planks, sticks, crafting table, wooden pickaxe, stone pickaxe, furnace, iron ingot (smelting), and iron pickaxe.
- **Mine (2 tasks):** Digging downward bare-handed to collect dirt, and with a wooden pickaxe to collect cobblestone.
- **Negative controls (4 tasks):** The same operations with a required precondition missing: crafting planks without logs, crafting a wooden pickaxe without planks, smelting without fuel, and mining stone without a pickaxe. The real environment always fails these, so world model predicted success is a hallucinated transition.
- **Composite (2 tasks):** Two-stage sequences: mining cobblestone then crafting a stone pickaxe, and smelting iron then crafting an iron pickaxe.

Task	Category	N	Real	World Model	Agreement	Optimism
Crafting Table	Craft	20	100%	100%	100%	0%
Furnace	Craft	20	80%	55%	55%	10%
Iron Pickaxe	Craft	20	80%	100%	80%	20%
Planks	Craft	20	100%	100%	100%	0%
Sticks	Craft	20	100%	100%	100%	0%
Stone Pickaxe	Craft	20	80%	100%	80%	20%
Wooden Pickaxe	Craft	20	80%	100%	80%	20%
Smelt Iron	Craft	20	80%	100%	80%	20%
Mine Dirt (Bare)	Mine	20	95%	55%	60%	0%
Mine Stone (Wooden Pick)	Mine	20	75%	40%	45%	10%
Mine $\rightarrow$ Craft Stone Pickaxe	Composite	20	35%	45%	90%	10%
Smelt $\rightarrow$ Craft Iron Pickaxe	Composite	20	20%	30%	90%	10%
Craft Planks (No Log)	Negative	20	0%	0%	100%	0%
Mine Stone (Bare)	Negative	20	0%	0%	100%	0%
Smelt (No Coal)	Negative	20	0%	0%	100%	0%
Wooden Pickaxe (No Planks)	Negative	20	0%	0%	100%	0%

Table 7: **World-model task evaluation.** Comparison of real-environment outcomes and world-model predictions across 16 tasks, with $N = 20$ trials per task. *Real* and *World Model* give task success rates in each setting; *Agreement* is the percent match of real and world model outcomes; *Optimism* is how often the world model overpredicts success (lower is better). Tasks are grouped into craft, mine, composite, and negative categories.

	FVD_{16} ↓			PSNR (dB) ↑		
Model (checkpoint)	Deployed	+ History guid. (×1.4)	Halves	Frame 1	Frame 8	Frame 16
WM-S	36.8	28.3	48.0	24.2	18.7	16.9
WM-B	30.4	25.0	43.0	24.7	18.6	16.7
WM-L	**24.0**	**21.2**	**37.2**	**25.4**	**19.5**	**17.6**

Table 8: **FVD Metrics.** Video prediction quality of the 64x64 world model output at three scales using an optimized checkpoint. FVD_{16} is computed over 16-frame rollouts for the deployed configuration and with history guidance at scale 1.4. The *Halves* column reports FVD under the half-split protocol, for which the real-vs.-real floor is 27.7. PSNR is measured at frames 1, 8, and 16 of the rollout to show error accumulation over the horizon. Larger models generally improve on metrics, with WM-L performing best.

Table 7 reports three quantities per task, each computed over N=20 paired trials in which the environment and the world model receive the same initial observation, starting inventory, and scripted action sequence. *Task success* is the fraction of trials in which the target item is obtained. *Agreement* is the fraction of trials in which the world model's predicted outcome matches the real outcome. *Optimism* is the fraction of trials in which the world model predicts success but the environment fails; the remaining disagreements are pessimistic errors, in which the model predicts failure on a trial that succeeds in the environment. We note that these scripted tasks are sequence-level, so these results additionally measure compounding error on outcome.

The world model agrees perfectly with the environment on planks, sticks, and crafting tables. Elsewhere its errors are task dependent and run in both directions. It is optimistic on four crafting and smelting tasks (wooden pickaxe, stone pickaxe, iron pickaxe, and iron ingot), predicting 100% success where the environment reaches 80%. It is pessimistic on furnace crafting (55% predicted vs. 80% real) and on both mining tasks (55% vs. 95% for dirt; 40% vs. 75% for stone). From analyzing attempts, we find that table placement failure is what affects these more complex crafting tasks. On the two composite tasks, real success rates are 35% and 20%, compared with predicted rates of 45% and 30%, respectively. Thus, the model is moderately optimistic on both composite tasks, with a 10-percentage-point gap between predicted and real success in each case. On all four negative controls, both real and predicted success are zero, giving perfect agreement, so the model does not produce transitions whose preconditions are missing. Overall, the model decently models the dynamics and is able to predict outcomes with sufficient accuracy.

	WM corrections	Real-env corrections
Attack share (%)	75.2	79.1
Place share (%)	0.2	0.2
Mean camera movement	0.88	0.75
Share of replay buffer (%)	21.8	28.3

Table 9: **Correction action distribution.** Action statistics of world-model and real-environment corrections. Corrections generated in the world model have action distributions similar to those executed in the environment.

The two error types affect training differently. Pessimistic errors cause false rejections, which reduce the number of accepted corrections without adding incorrect supervision. Optimistic errors cause false acceptances and are the ones that can place incorrect labels in the replay buffer. The optimistic errors we observe concentrate in macro-driven crafting. There, the executor has already checked recipe preconditions, so we hypothesize that a false acceptance usually still pairs a valid state with an appropriate action program, and the real-environment failure stems from execution rather than the plan. Aggregate action statistics from generated corrections are similar across backends (Table 9): attack actions make up 75.2% of world-model corrections and 79.1% of environment corrections, and mean camera movement is 0.88 versus 0.75. Hence, overall aggregate action distribution similarity can be one reason for successful downstream training, though the marginal cannot be guaranteed.

B Training Details

B.1 Baselines

This section details the reference methods we use in our primary performance comparison. To our knowledge, no existing post-training method for MineRL uses a comparable combination of

VLM guidance, action primitives, and a learned world model, so we report reward-based RL as a reference point rather than a same-information baseline. We do not compare against human-in-the-loop corrective methods such as DAgger, HG-DAgger or Hi-WM, as collecting human corrections for every round of online training (60 rounds, with corrections up to 1,200 ticks at 20 Hz) is impractical at our scale. Removing this requirement and automating the corrective learning loop is a central motivation for ReVision.

PPO (real environment). We use a standard PPO implementation (Schulman et al., 2017), following CleanRL (Huang et al., 2022), for a 130-way categorical policy. Rewards pass through a running normalizer, and the bootstrap value at truncated episodes is applied in normalized space.

Following VPT (Baker et al., 2022), we add an analytic KL penalty toward a frozen copy of the BC policy used for initialization, with its coefficient annealed over training. The KL is computed over the 130-way action logits, so it does not depend on how actions are decoded. The actor-critic reuses the BC backbone, so its policy logits exactly match the BC policy at initialization.

Dreamer-style imagination training (world model). This baseline adapts the DreamerV3 imagination objective (Hafner et al., 2025a) to our world model. The actor is trained with REINFORCE using a value baseline and an entropy bonus. The critic regresses onto λ-returns, regularized toward a slow EMA copy of itself. Following DreamerV3, advantages are scaled by a running estimate of the 5th-to-95th percentile range of returns. A KL anchor to the frozen BC policy keeps the actor from exploiting states the world model predicts poorly.

The environment is the frozen 406M-parameter mmDiT world model. Milestones are read from the world model's predicted inventory. Imagined rollouts start from frames sampled at random from the decoded human dataset. We use similar imagination horizon as the V3 implementation. This baseline uses no real environment interaction.

Open-loop VLM planner. This baseline uses the same VLM planner and macro-action interface as the ReVision corrector, with the world model removed entirely. Proposed actions are executed directly, without world-model rollouts or screening. The corrector's acceptance-gate instructions are replaced with a plain objective for the full episode. The planner observes the current frame, emits a batch of macro actions, executes them in the real environment, and then observes again. A new query is issued only once the current batch of actions and intents has been used up. Each query also includes the starting inventory and the net change in inventory since the start of the episode, because the VLM tracks inventory unreliably from pixels alone. Episodes are capped at 1,500 VLM calls.

B.2 HYPERPARAMETERS

All BC CNN arms are initialized from the same behavior-cloning checkpoint, and the VLA arm from its supervised fine-tuning checkpoint. Tables 10–14 list the training and hyperparameter configurations. Note that in training, episode rollouts are often shorter than the maximum episode step length.

Setting	Value
Data collection and correction	
Rounds N	60
Rollouts per round	8 episodes
Episode length	18,000 ticks maximum
Chunk length / stride	600 / 600 ticks
Candidates ranked per episode k	6
Corrections per episode	1
Attempts J	3
Correction horizon H	1,200 ticks
Selection threshold	0.5
Acceptance threshold	0.5
Candidate re-draw decay	0.99
Policy update	
Optimizer	Adam, $\epsilon = 1.5 \times 10^{-4}$, weight decay 10^{-5}
Learning rate	6.25×10^{-6}
Schedule	no warmup, cosine to $0.1\times$
Batch size	32
Epochs per round	1
Gradient clipping	1.0
Correction oversampling	$4\times$
Demo : correction ratio	$\approx 1:1$ per batch
Replay decay λ	0.9
Augmentation	horizontal flip

Table 10: ReVision hyperparameters used in training.

Setting	Value	Setting	Value
Parallel envs	8	Rollout length	256 steps
Total env steps	30M	Max episode length	18,000 steps
Discount γ	0.99	GAE λ	0.95
Clip ϵ	0.1	Value clip	0.2
Epochs per update	4	Minibatches	8
Learning rate	10^{-4} (annealed)	Max grad norm	0.5
Entropy coef.	0.01	Value coef.	0.5
KL-to-BC coef.	$0.2 \rightarrow 0.1$	Target KL	0.03

Table 11: PPO hyperparameters.

Setting	Value	Setting	Value
Imagination horizon	12 WM steps (36 ticks)	Batch size	16
DDIM steps	10	WM context frames	8
Optimizer	AdamW	Learning rate	3×10^{-5}
Warmup	50 updates, then cosine	Max grad norm	0.5
Epochs per update	1	Minibatches	4
Discount γ	0.99	λ	0.95
Entropy coef.	$3 \times 10^{-3} \rightarrow 3 \times 10^{-4}$	KL-to-BC coef.	$0.3 \rightarrow 0.1$
Value coef.	0.5	Slow-critic reg.	1.0
Slow-critic EMA	0.98	Return-norm EMA	0.01

Table 12: Dreamer-style REINFORCE hyperparameters.

B.3 VLM USAGE AND COST

Arm	Scorer	Planner	Meta
ReVision GPT	`gpt-5.6-terra`	`gpt-5.6-luna`	`gpt-5.6-terra`
ReVision Gemini	`gemini-2.5-pro`	`gemini-2.5-flash`	`gemini-2.5-pro`

Table 13: VLM models used by each corrector arm. For cost purposes, we do not use GPT-5.6-Sol and Gemini-3.1-Pro.

				Corrections			
Arm	Iter	Calls	Calls/iter	Attempts	Accepted	Turns/attempt	Input tokens
ReVision-E GPT	60	2,744	45.7	471	197 (41.8%)	4.1	≈20M
ReVision-E Gemini	60	7,592	126.5	700	231 (33.0%)	9.3	≈50M

Table 14: VLM usage per corrector arm. An episode yields a correction problem only if the scorer rates at least one chunk at priority ≥ 0.5; in the GPT arm, 278 of 480 episodes did, so attempts can be fewer than episodes. Acceptance is reported per attempt; per problem, the GPT arm accepts 197/278 (70.9%). The Gemini arm uses more calls mainly because it takes more planner turns per attempt (9.3 vs. 4.1), and secondarily because it makes more attempts. Per-call context is comparable. Call counts and correction outcomes are exact; token totals are approximate.

Method	Env. steps	VLM calls	VLM input tokens	Policy train GPU-h	Total GPU-h	GPU
ReVision-E (GPT)	6.1M	2,744	≈20.0M	3.6	12.7	H200
ReVision-E (Gemini)	5.9M	7,592	≈50.0M	5.0	25.3	H100
ReVision (GPT)	6.2M	5,318	≈32.9M	5.0	13.3	H200
ReVision-W (GPT)	0	3,301	≈22.0M	4.5	64.2	H100
PPO (6M)	6.1M	0	0	13.8	13.8	H100
PPO (30M)	30.0M	0	0	71.0	71.0	H100
Dreamer-style RL	0	0	0	≤4.2	102.1	H100

Table 15: **Compute and interaction accounting.** Computed for BC-initialized runs. Env. steps are real MineRL ticks, including real-environment corrections for ReVision-E. VLM calls count chunk-scorer, meta-scorer, planner, and progress-judge API calls; input tokens exclude output and reasoning tokens, and API-side VLM compute is not included in GPU-hours. Policy training counts data preparation and optimizer updates for ReVision arms, and the full job for PPO, where environment stepping and updates are interleaved. GPU-hours are not normalized across hardware.

B.4 RLBENCH EXPERIMENTAL PROTOCOL

Policy and inputs. We fine-tune the released RVT-2 checkpoint. The student's observations are unchanged from upstream RVT-2: RGB-D images from four cameras (front, left shoulder, right shoulder, and wrist) and proprioception.

Training. Student rollouts are collected in the environment, disjoint from the released 25-episode test panel. After a student failure, a VLM teacher (GPT-5.6, high reasoning effort) takes control at a root state and issues end-effector keyframes. Corrections are rolled out in a state MLP world model. The world model outputs state, and the RLBench environment renderer produces RGB frames. Each accepted correction is stored as a root row at the takeover state followed by downstream rows, in the same replay format as the demonstrations. Collection differs across tasks. Each batch of 64 contains 48 rows from the 100 released training demonstrations per task (for Stack Blocks, the RVT authors' released training replay), 8 correction roots, and 8 downstream rows, sampled uniformly within each pool. A KL penalty on demonstration rows keeps the coarse-stage translation heatmap close to the frozen pretrained RVT-2.

RVT-1 and comparison with RACER. To compare with RACER, which builds on RVT-1, we also apply ReVision to the released RVT-1 checkpoint, using the upstream PyTorch3D renderer. RVT-1 has a single translation stage, so no KL penalty is used. Training runs three online rounds of 120 rollouts (192 for Stack Blocks); each round rolls out the latest policy, aggregates corrections from all rounds, and continues training from the previous round's weights. Place Cups and Stack Cups were stopped after two rounds for time. Evaluation uses 25 actions and 5 repeats for every task, following the RVT row of RACER's Table I.

B.5 METAWORLD EXPERIMENTS

We then evaluate four MetaWorld v3 tasks (Yu et al., 2020): soccer, basketball, push back, and pick-place. These four tasks are more difficult than others like button press, door close, and are unsaturated with pixel BC. The policy uses a vision encoder followed by an MLP and conditions on the RGB frame and task state. For each task, we initialize the policy with pixel BC trained on 200 and 250 demonstrations, then apply ReVision on the 200 demonstration policy during post-training. We train three seeds and evaluate each on the same 50 fixed worlds, which are disjoint from the training worlds. Success means achieving the task at any point during a 500-step episode.

Method	Soccer	Basketball	Push back	Pick-place
Pixel BC (200 demos)	43.3 ± 0.7	57.3 ± 4.4	64.0 ± 4.0	75.3 ± 7.1
Pixel BC (250 demos)	41.3 ± 4.4	65.3 ± 1.8	65.3 ± 4.1	82.7 ± 1.8
ReVision (VLM corrector)	50.0 ± 2.0	68.0 ± 2.0	72.0 ± 3.1	80.7 ± 2.9
Oracle corrector	50.7 ± 1.8	68.0 ± 2.3	71.3 ± 6.7	87.3 ± 4.1

Table 16: **MetaWorld success rates.** Values are percentages, reported as mean $\pm$ across-seed SEM over three seeds, each evaluated on 50 worlds. Pixel BC is evaluated twice after training with 200 and 250 demonstrations per task. ReVision and oracle trained starting from 200 demo pixel BC.

We train three independent seeds per task. Each seed is evaluated on the same fixed set of 50 worlds, disjoint from the training worlds. An episode lasts at most 500 steps, and success is recorded if the task is achieved at any point. Table 16 reports the mean success percentage and across-seed SEM. The restored-state rejection-and-retry procedure is illustrated in Fig. 5.

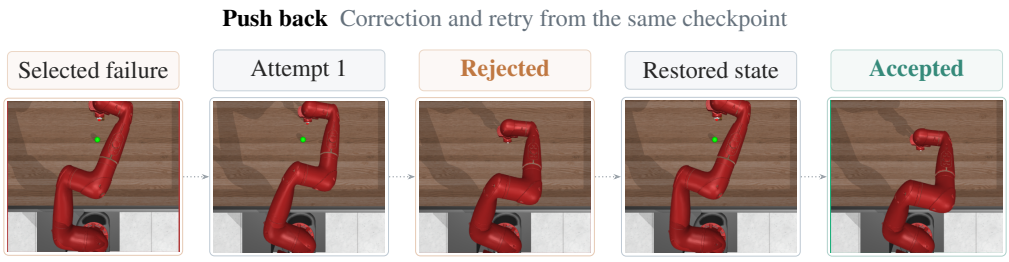


Figure 5: Real-environment sample of MetaWorld learning. Push-back correction from a restored state: the selector rejects the first attempt and accepts a retry.

B.6 DIAGNOSTIC AND SELECTION EXAMPLES

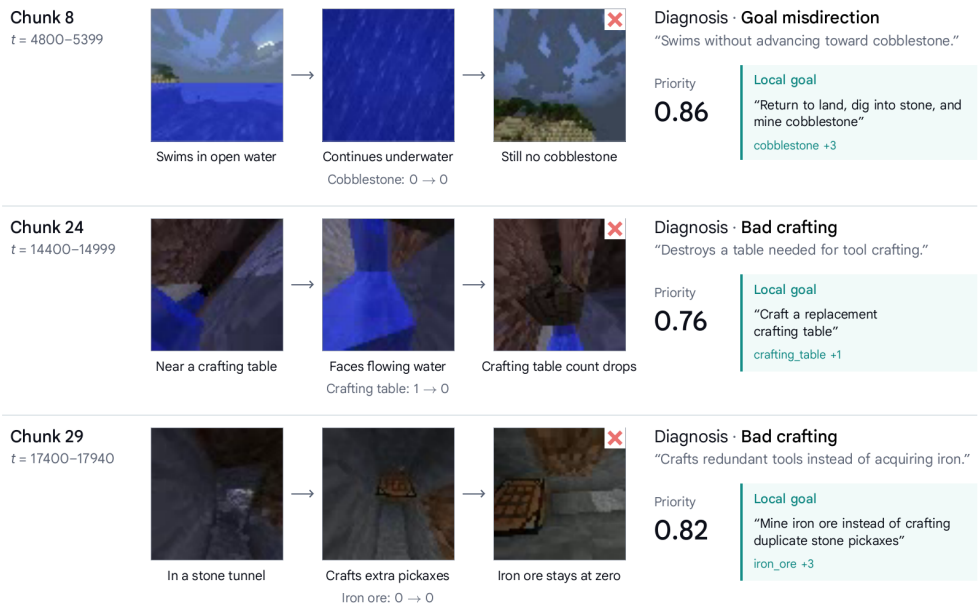


Figure 6: **Failure scoring along one rollout.** Selected chunks evaluated by a VLM from a sample rollout. Each chunk is represented by three frames in the figure, with the VLM-assigned priority score, reasoning, and local goal displayed to the right.

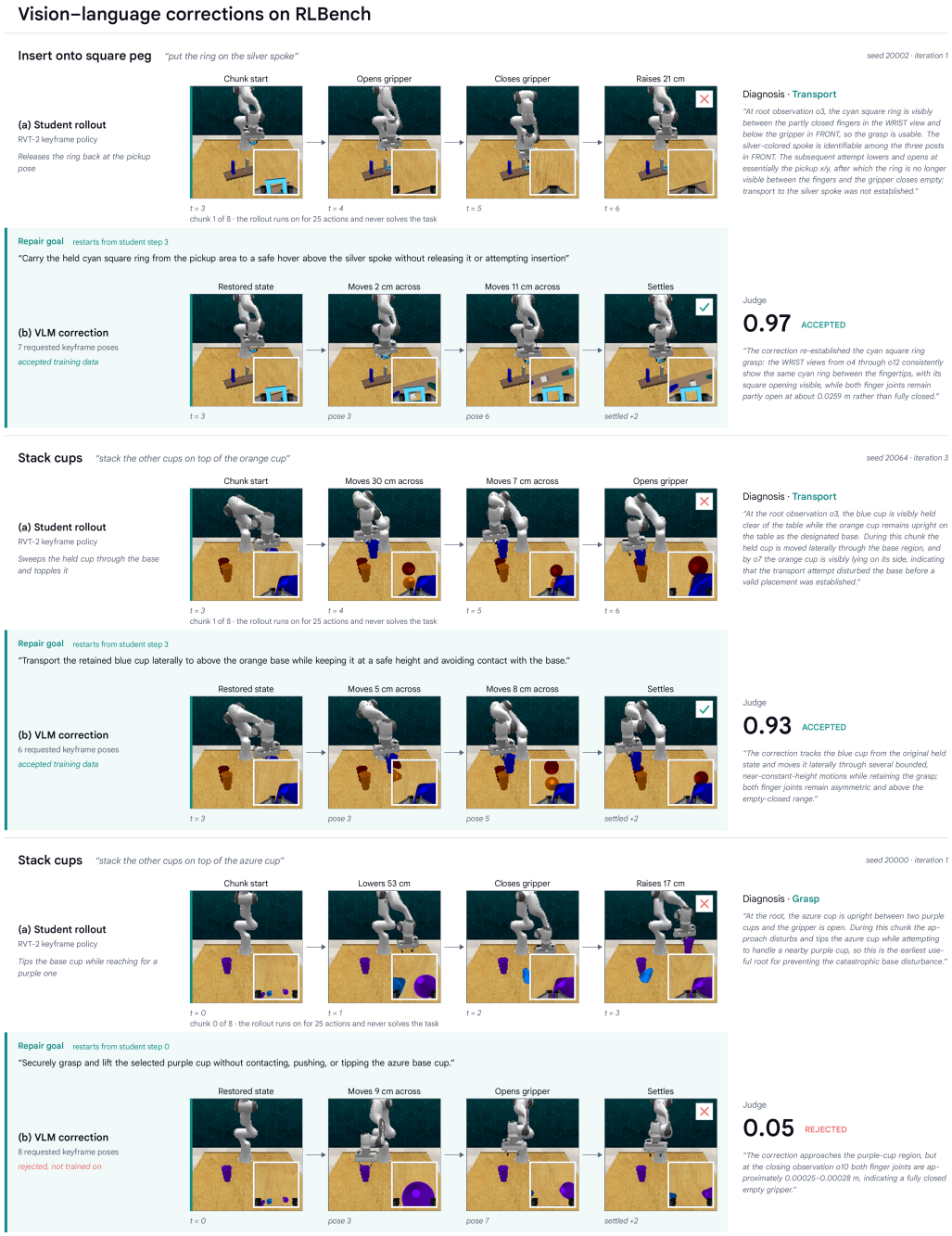


Figure 7: **Correcting a keyframe policy from vision and language.** Three episodes, one per block. **(a)** The policy's attempt at the chunk the scorer selected, each frame titled with the keyframe pose requested there and the diagnosis quoted at right. The blue band shows the repair goal the scorer attached to that chunk. **(b)** The correction executed after restarting from that same state, ending in the judge's score and verdict. Only corrections scoring at least 0.85 are trained on.

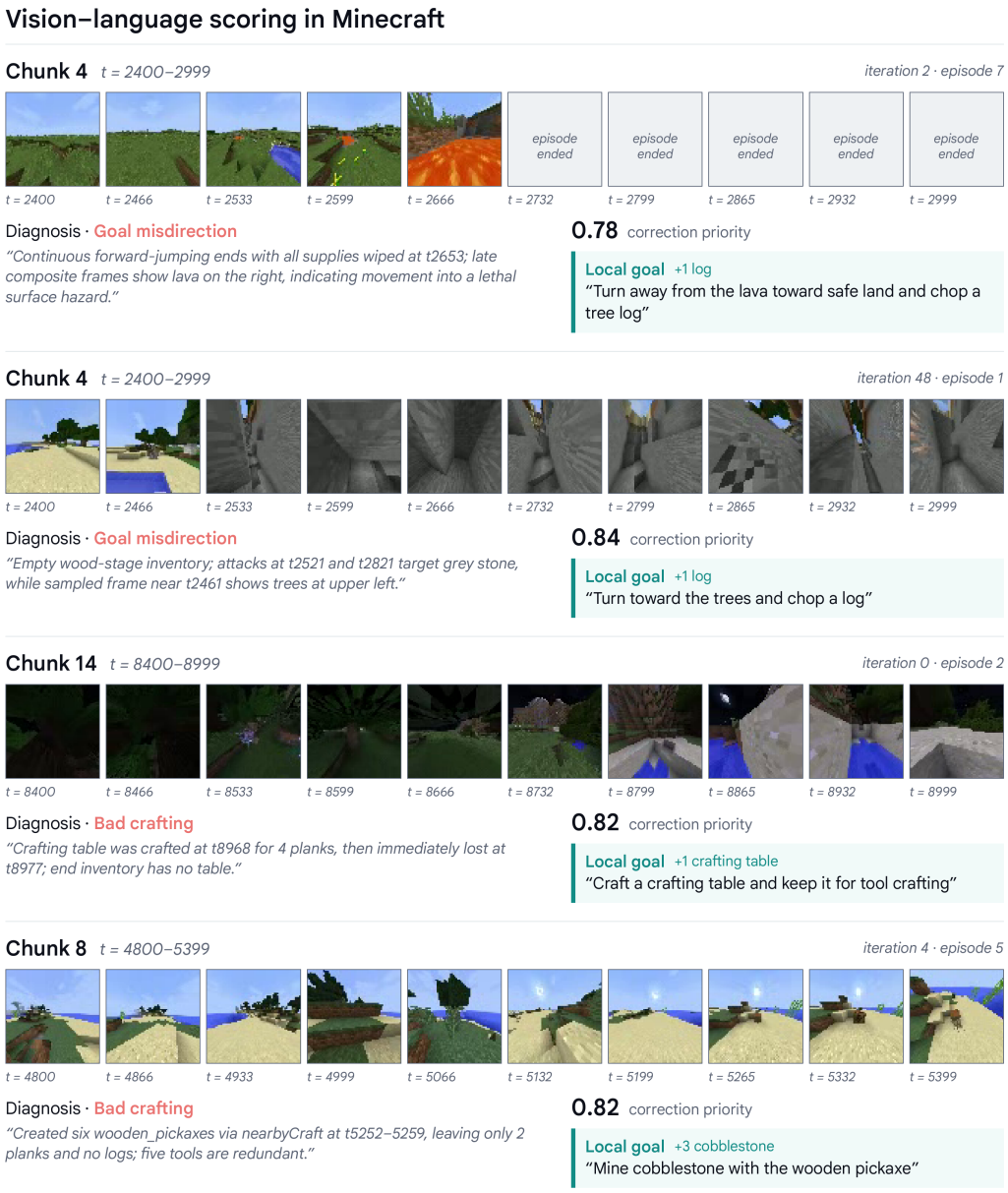


Figure 8: **Scoring Minecraft rollouts with the VLM.** Four chunks flagged by the scorer, one per block, spanning the wood, crafting-table and tool stages. Each block shows the 10 evenly spaced frames of a 600-tick chunk exactly as the scorer receives them; greyed tiles fall after the episode ended. Below, the scorer's failure diagnosis and its quoted evidence (left), and the correction priority in $[0, 1]$ with the local goal and target inventory change it hands to the corrector (shaded band, right). Unflagged chunks keep priority 0 and are never corrected. Annotations are VLM judgments, not ground truth.

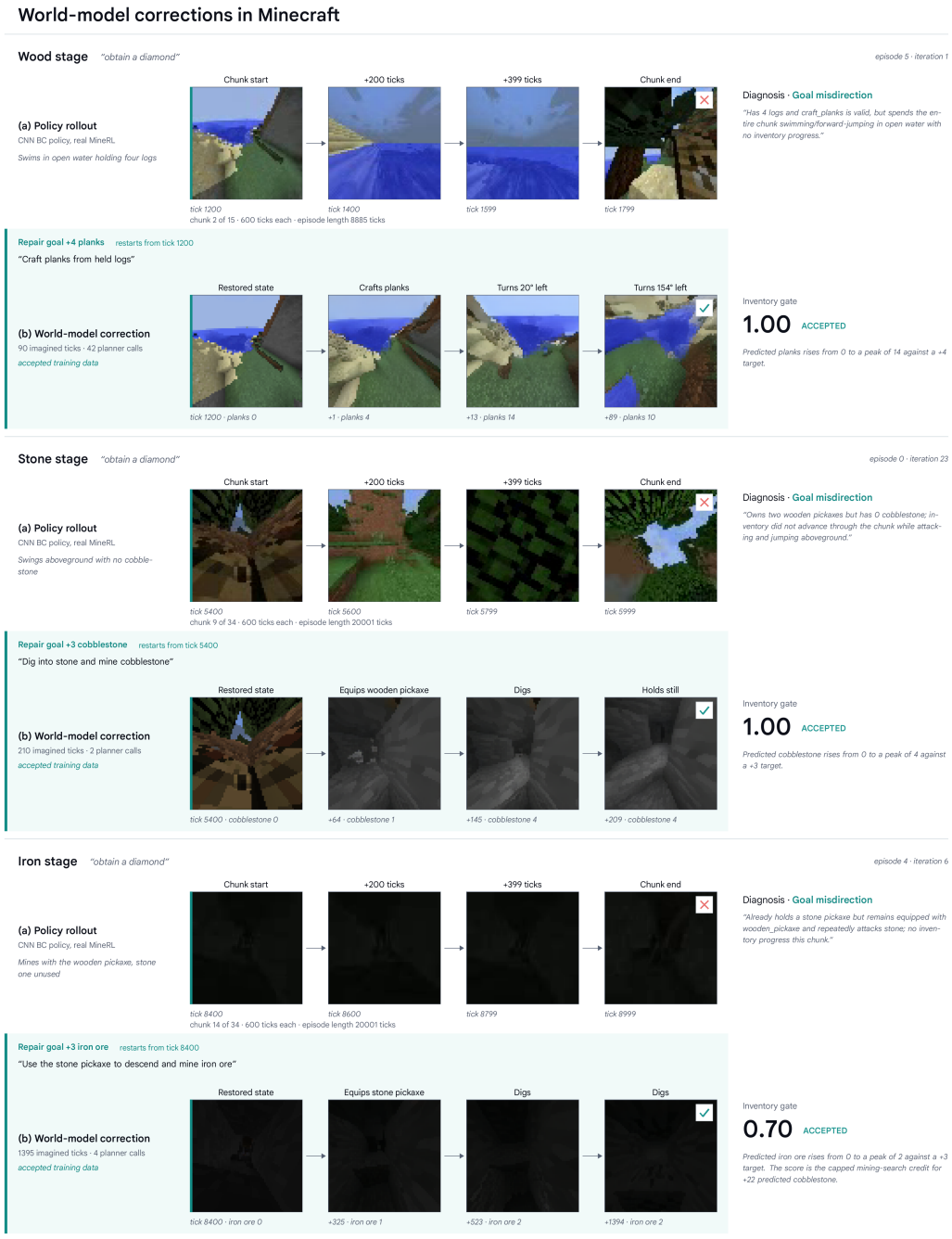
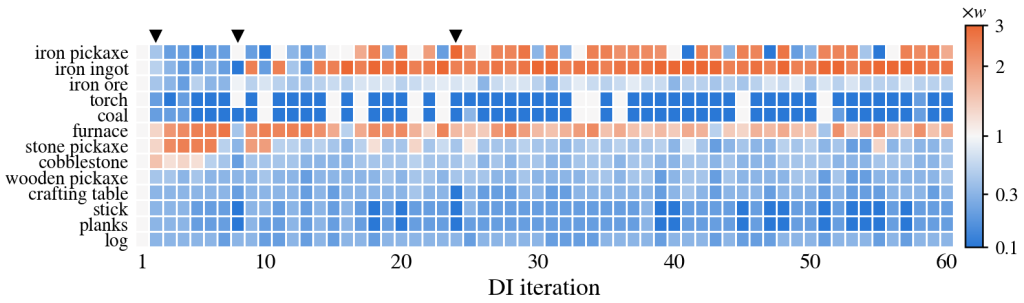


Figure 9: **Correcting a Minecraft policy inside the world model.** Three episodes, one per block. **(a)** The policy's real rollout over the 600-tick chunk the scorer selected, with the diagnosis quoted at right. The blue band shows the repair goal the scorer attached to that chunk. **(b)** The correction imagined by the world model from the first frame of that chunk, each frame titled with the planner's action there and captioned with the predicted count of the goal item, ending in the correction's score. Scores and inventory changes are world-model predictions. Only corrections scoring at least 0.25 are trained on.



Iteration 2

"The bottleneck is the scarce post-wood transition: episodes reach a wooden pickaxe reliably, but many do not convert that progress into stone-tier resources and stationary stone-stage progression. Wood gathering and basic crafting are already largely solved; stop spending correction budget on them despite their easy accepts. Do not chase coal, torch, buried ore, or death states as correction targets: deaths are a survival capability problem rather than a per-chunk teachable failure."

Iteration 8

"The bottleneck is the sharp transition from completing the stone-tool/furnace tier to reaching deep buried-resource progress: only 2 of 23 episodes that clear both stone-tier requirements reach iron ore. Wood, cobblestone, and the stone crafting tier are already reliably cleared and should not consume correction budget. Treat easy-looking buried-ore accepts as suspect proxy credit; prioritize genuine frontier states, while not chasing post-ore progression that the unaided policy almost never reaches."

[capability] Robust survival during descent/mining: 62% of episodes end in death, which cannot be repaired by short mid-episode corrections.

Iteration 24

"The bottleneck is late iron progression: the large unaided drops are from the completed stone tier into ore, then into a live smelt and final deterministic tool transition; prioritize scarce late-stage stationary prerequisite and crafting states. Wood through cobblestone and the initial stone-tool transition are largely solved, so do not spend correction budget on respawn-driven low-tier repetition. Treat buried-ore correction accepts as suspect proxy credit and do not chase ore acquisition as though it were a reliably short-correctable failure."

Figure 10: Meta-scorer priorities and bottleneck summaries across iterations, reflecting model judgments.

C ACTION PRIMITIVES

A vision-language model cannot act at environment rate (20 Hz keyboard/mouse input in Minecraft, 500 end-effector commands per MetaWorld episode), so in both environments the teacher emits a short *program* of named, parameterised primitives. Each primitive expands deterministically into native actions that are executed and checked in the simulator; the resulting (o_t, a_t) pairs enter the behaviour-cloning buffer in the student's native action space. The student never sees the primitives themselves.

C.1 MINERL

Action space and timing. Each primitive step sets a factored MineRL action: eight binary buttons (movement, jump, sneak, sprint, attack), a relative camera rotation discretised into 41 bins per axis over $\pm 25^\circ$, and categorical `place`, `equip`, `craft`, `nearbyCraft` and `nearbySmelt` heads. Primitives use two camera step sizes, $\approx 6^\circ$ and $\approx 12^\circ$. Because the camera is relative-only, Malmo's $\pm 90^\circ$ pitch clamp is the only absolute reference, and several primitives over-rotate into it and then tilt back a fixed number of steps to reach a known pitch from any starting angle ("clamp slam"). Durations are calibrated in world-model steps of three game ticks; in the real environment (one tick per step) the place-and-craft macros use correspondingly longer budgets.

Planner vocabulary. Table 17 lists the primitives the planner may emit. The focus is to provide an interface for the VLM planner, and we focus on providing a wrapper around existing actions and a spatial aim interface for the VLM. Note certain actions like crafting and smelting are already single categorical actions (craft, nearbyCraft, nearbySmelt). The two spatial primitives map a pixel on the 64×64 frame to a camera rotation (assuming a 70° field of view) and are the planner's means of visual servoing. The executor and scripted correctors additionally use a few internal primitives (e.g. `nudge`, `unstuck`, `reset_view`) that the planner cannot call.

primitive	behaviour
`forward(blocks, jump_pct)`	walk forward 4 steps per block, optionally jumping; capped at 5 blocks so the planner re-aims regularly
`swim(blocks)`	forward with jump held, so the agent surfaces rather than drowns
`turn_left/right(deg)`, `look_up/down(deg)`	rotate the camera in $\approx 6^\circ$ steps
`left_button(steps)`	hold `attack` at the crosshair
`dig_down(n)`	pitch straight down, mine through $n \leq 12$ blocks (falling onto each), re-level
`dig_tunnel(n)`	level the camera, then for each of $n \leq 8$ cells mine the eye- and foot-level faces and step in
`equip(item)`, `craft_planks`, `craft_sticks`, `craft_crafting_table`, `craft_torch`	single-step equip or inventory craft
`craft_*_pickaxe_recover`, `craft_*_axe_recover`, `craft_furnace_recover`	place a crafting table, craft the item (* = wooden, stone or iron; tools are also equipped), break and re-collect the table
`smelt_iron_recover(n)`	as above with a furnace, smelting $n \leq 9$ ore
`smelt_and_upgrade_pick`	smelt 3 ore, then craft and equip an iron pickaxe
`place_torch`	place a torch; on failure, retry once against a wall
`spatial_aim(target_pixel)`	rotate the camera to bring the pixel to the crosshair
`spatial_interact(target_pixel movement, jump, click)`	aim, then hold the requested keys for 5 steps

Table 17: **MineRL primitives available to the planner.** The two `spatial_*` primitives are emitted as JSON objects; all others as `name(param)` calls.

From primitives to student labels. The student is trained on the per-tick actions the executor produces, not on the primitives. Each primitive expands into a per-tick MineRL action, with camera

bins converted to degrees (a large turn is $12.2°$ per tick) and craft, equip, place, and smelt slots converted to MineRL commands. For ReVision-E, these actions are executed in the environment; for ReVision and ReVision-W, the same actions condition the world model and are stored with its predicted frames. In both cases, the resulting trajectory is saved in the MineRL demonstration format, with correction ticks flagged, and labeled by the same pipeline used for $\mathcal{D}_0$: camera deltas are summed over a 3-tick window, and each tick is mapped to one of 130 joint actions, comprising 113 combinations of up to three buttons with an optional $\pm 22.5°$ single-axis camera move and 17 dedicated craft, equip, place, and smelt actions. The PPO baseline uses the same 130-way action space. Because demonstrations and corrections share this mapping, corrections introduce no representability gap beyond that already present in the demonstration data. On correction data, the mapping preserves all button states, all craft, equip, place, and smelt actions, and 99.2% of camera directions. The main distortion is camera magnitude: a large turn summed over the window ($36.6°$) is labeled $22.5°$, which changes turning speed but not direction.

Preconditions and verified placement. Before a craft or smelt macro runs, the planner checks the inventory against the recipe (plus a crafting table or furnace, and coal or planks as fuel for smelting) and drops calls that cannot succeed. Because a macro can still fail silently when its block cannot be placed, the next observation is checked against the macro's expected product. After each attempt an executor-side check compares the item's inventory count. On success, execution skips ahead to the craft or smelt step, and if every attempt fails the craft and break phases are skipped.

C.2 METAWORLD

Action space and controller. The Sawyer action is a 4-vector in $[-1, 1]^4$: an end-effector displacement and a gripper command (-1 open, $+1$ closed). MetaWorld's scripted experts are waypoint programs over a single proportional controller, $a_{xyz} = \mathrm{clip}\big(p\,(x^* - x_{\mathrm{hand}}),\ -1,\ 1\big)$. The primitive library (Table 18) exposes exactly this controller, so the scripted oracle and the VLM teacher share one action space and the oracle is an upper bound on what a program can achieve. Primitives are closed-loop: the action is recomputed from the observed hand position at every step, and a primitive ends when the hand is within tolerance of its target or a step cap is reached.

primitive	behaviour
`move_to`	drive toward an absolute target x, y, z
`move_by`	drive toward the entry hand position plus a delta
`move_to_landmark`	track the currently observed `object` or `goal` plus an offset, re-read every step so the hand follows an object moving under contact; `hold_z` freezes the target height for horizontal pushes
`set_grip`	hold position and drive the gripper to a value for a fixed number of steps

Table 18: **MetaWorld primitives.** Motion primitives take a grip value, gain $p \in [1, 40]$ (default 25), tolerance $\in [0.005, 0.08]$ m (default 0.02) and step cap $\in [1, 160]$ (default 60); bounds are enforced by the parser.

Programs. The teacher returns strict JSON containing a one-sentence plan summary and a list of 1–8 primitives, executed for at most 500 environment steps. Malformed programs are rejected at parse time and cost the teacher a retry rather than executing as a no-op. A program enters the buffer only if an independent quality judge scores the trajectory at least 0.70, which filters out mistake-then-recovery loops and contradictory actions; this applies to bootstrap demonstrations and corrections alike.

Task conventions. The primitives are task-agnostic. The only task-specific input to the teacher is a short manual per task naming the physical contact point and the gains and grip values used by the scripted experts (Table 19).

C.3 RLBENCH

Action spaces. The primitives below are our own interface rather than part of RLBench. They are executed through two native action spaces. The incremental Panda controller takes a 7-vector $[\Delta x, \Delta y, \Delta z, r_x, r_y, r_z, g]$ (metres, radians). The RVT-2 keyframe controller takes an absolute world

task	convention
button-press-topdown	`object` is the button contact point, `goal` its depressed position; align above, then descend tracking the button. Grip $+1$, $p = 25$.
door-close	`object` is a door-body reference; the handle is at `object` $+ [0.05, 0.12, 0.10]$. Approach from above, descend to the handle, push toward `goal`. Grip $+1$, $p = 25$.
push-back	`object` is the puck centre; descend with neutral grip, then push to `goal` at constant height (`hold_z`). Grip $+0.9$, $p = 10$.
pick-place	open, descend onto the object, close for 14 steps, carry to `goal`; never track the object after grasping. $p = 10$.

Table 19: **Per-task conventions in the MetaWorld teacher manual.**

pose with an $xyzw$ quaternion, a binary `gripper_open` flag and an `ignore_collisions` flag; the arm motion planner produces the path between keyframes. Gripper signs differ between the two: waypoint programs use ± 1 (snapped from $[-1, 1]$, positive opens), native RVT-2 uses $1/0$.

Controller. Unlike MetaWorld's proportional controller, translation is a bounded Cartesian step, $a_{xyz} = d \cdot \min\big(1,\ v/\|d\|\big)$ with $d = x^* - x_{\text{hand}}$ and speed v, recomputed from the live hand position at every step. The waypoint vocabulary fixes $r_x = r_y = 0$, so the gripper stays top-down and only yaw is controlled. Yaw error is wrapped by a symmetry period (90° for cube faces, a full turn when orientation must be unique) and clipped to 0.30 rad per step.

primitive	behaviour
`move_to`	drive toward an absolute target x, y, z
`move_by`	drive toward the entry hand position plus a delta
`rotate_yaw`	hold position and rotate toward an absolute yaw, modulo the symmetry period
`set_grip`	hold position and actuate the gripper for a fixed number of steps (default 3, closed); needed when the grip must change at an already-reached target, since motion primitives check completion before acting

Table 20: **RLBench waypoint primitives.** Translation primitives take a grip value, speed $\in [0.005, 0.03]$ m/step (default 0.03), tolerance $\in [0.003, 0.05]$ m (default 0.01) and step cap $\in [1, 120]$ (default 40). `rotate_yaw` takes a yaw $\in [-2\pi, 2\pi]$, symmetry period $\in [0°, 360°]$ (default 90°), tolerance $\in [0.01, 0.5]$ rad (default 0.05) and step cap $\in [1, 40]$ (default 15). Bounds are enforced by the parser.

Programs. The teacher returns JSON containing 1–12 primitives, executed for at most 300 environment steps. The parser rejects unknown names, malformed vectors and out-of-range parameters, but does not check geometric feasibility. A primitive ends at tolerance or at its step cap; reaching the cap neither certifies arrival nor aborts the program. Each executed step is recorded as a pre-action observation paired with its action. Primitive completion, task success and acceptance of a correction are treated as separate outcomes, and a correction enters the buffer only after passing the task's acceptance checks.

Keyframe execution. For the RVT-2 student, each waypoint primitive is resolved to a target pose and handed to the motion planner, so speed and step parameters are omitted and paths need not be straight. The grip command is applied only after the pose is reached: a descent with a close command closes at the bottom, and a carry with an open command releases at the destination, so placement programs release at the placement pose before a separate withdrawal. The executor may merge compatible alignment and grip operations and insert clearance waypoints, which count against the action budget.

Visual teacher. A second, visual-only teacher sees RGB and proprioception and emits commands that each compile to one native keyframe (Table 21). Pixel commands use normalised image coordinates and camera calibration only (no detection, segmentation, depth or object poses). They move horizontally by at most 0.12 m, preserve height and orientation, and always use collision-checked planning. They never descend, rotate, grasp or release, so those steps are issued as pose commands, with fresh observations closing the loop.

command	behaviour
absolute pose	full native keyframe; the quaternion is normalised
`relative_pose`	translation (≤ 0.12 m) and axis-angle rotation ($\leq 45^\circ$) from the measured gripper pose, in world or tool frame
`tabletop_pixel_servo`	the same tabletop object marked in ≥ 2 views; camera rays are intersected with the table and the median xy is taken; for coarse alignment above loose objects
`multiview_pixel_servo`	a destination point marked in all three fixed views and triangulated; for transport toward elevated destinations at safe clearance
`multiview_offset_servo`	held-object and destination points marked in ≥ 2 matching views; moves by their horizontal difference, correcting for an object that is off-centre in the gripper

Table 21: **RLBench visual-teacher commands.** Pixel commands (lower block) move horizontally only, capped at 0.12 m.

Task conventions. As in MetaWorld, the primitives are task-agnostic. The only task-specific input to the teacher is a short manual per task, for `stack_blocks`, `stack_cups`, `place_cups`, `insert_onto_square_peg` and `place_shape_in_shape_sorter`; the visual teacher receives visual manuals instead.