ASSAY: RECOVERING FILTERED ENVIRONMENTS WITH VERIFIER-GROUNDED ACTOR–CRITIC RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Code-derived reinforcement-learning environments are often selected using a screening model's rollout outcomes. Such selection mixes two questions: whether an environment is reliable, and whether a particular training recipe can use it effectively. We study this distinction in CODEMIDAS, where an all-pass/all-fail screen removes 2,628 of 8,173 tasks that have passed execution-consistency, leakage, and verifier-agreement checks. A unanimous training group gives GRPO zero reward-driven advantage, but a unanimous screening group does not establish that future training groups will be degenerate. We introduce ASSAY, which re-admits these tasks, uses single-rollout asynchronous actor–critic optimization, and pretrains the critic on outcome-labeled records retained during environment construction. On the retained RL task pool, ASSAY improves MiMo-V2.5 from GRPO's 54.4 to 56.1 on SWE-bench Pro and from 21.7 to 24.3 on DeepSWE. Expanding the pool raises these scores to 57.9 and 27.5. The same expansion changes CODEMIDAS Val by $+2.7$ percentage points under ASSAY and -0.4 under the evaluated GRPO configuration. Results on GLM-5.3-Flash show the same direction of improvement. Ablations, prompt-exposure controls, and training-cost measurements characterize the roles of value initialization, sampling, and asynchronous execution. These results support a practical conclusion: rollout screening should not be treated as an optimizer-independent verdict on training-data value.

1 INTRODUCTION

Reinforcement learning on executable software tasks requires both diverse environments and trustworthy rewards (Wei et al., 2025; Chen et al., 2026a; Zeng et al., 2026). Recent pipelines construct this supervision from repositories, development artifacts, or existing tests (Jain et al., 2025; Yang et al., 2025; Fu et al., 2026). CODEMIDAS takes source code as its only task-specific input and produces 5,545 training tasks from 3,185 codebases in 23 languages (Ye et al., 2026). Each task combines a behavioral specification, a development environment, and a hidden executable verifier.

The final selection stage introduces a different kind of judgment. After checks for execution consistency, residual leakage, and agreement between verifier verdicts and the task statement, a screening model attempts each task. Only tasks with both passing and failing attempts are retained. This removes 2,628 of the 8,173 tasks that reach the final screen. Unanimous outcomes can indicate extreme difficulty, an underspecified statement, or weak tests; the screen alone does not distinguish these possibilities. Its output therefore reflects the screening policy and sampling budget, not just an intrinsic property of the environment.

Screening is related to optimization, but is not its equivalent. Under binary rewards, a group-relative estimator assigns zero reward-driven advantage to every rollout in an all-pass or all-fail training group. Avoiding such groups is a reasonable efficiency goal. However, a small offline screening group does not predict future group degeneracy exactly. Even under the simplifying assumption of an unchanged policy with task success probability $p = 0.8$, four attempts are unanimous with probability 41.12%, whereas a group of 32 is unanimous with probability 0.079%. Different screening and training policies introduce an additional distribution shift. Tasks discarded by this heuristic can therefore remain useful even to GRPO; whether a larger pool helps a particular learner is an empirical question.

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.

A construction pipeline also produces value supervision. Actor–critic training offers a complementary way to use the pool. A state-dependent value function can provide different advantage estimates along a trajectory, without requiring sibling rollouts to disagree. The challenge is to learn a useful critic: terminal outcomes alone do not identify which intermediate actions made progress. Code-derived environments provide reference-solution records, successful and failed agent attempts, and adversarial leakage traces. These records supply an initialization signal that is absent from a cold critic, while subsequent RL updates adapt value estimates to the evolving policy.

We combine these observations in ASSAY. The method keeps the construction and quality-checking stages of CODEMIDAS, re-admits the final screen's rejected pool, and uses verifier-grounded value pretraining (VGVP) with Single-rollout Asynchronous Optimization (SAO; Hou et al. 2026). We adopt SAO's optimizer rather than introduce a new policy-gradient estimator. Our focus is the interface between task selection, construction-time supervision, and downstream learning.

Contributions.

- **A diagnosis of rollout-outcome filtering.** We distinguish environment reliability, finite-sample screening outcomes, and policy-dependent training-group degeneracy, and quantify their mismatch (§3).
- **A data-recovery and value-initialization recipe.** We recover 2,628 screened-out environments and reuse construction-time outcome records for critic pretraining, while retaining prior quality checks and an online leakage guard (§4).
- **Controlled empirical evaluation.** We compare RL task pools under GRPO and ASSAY, evaluate component ablations on MiMo-V2.5, replicate the recipe on GLM-5.3-Flash, and report prompt-exposure and training-cost controls (§5).

2 BACKGROUND

2.1 CODE-DERIVED RL ENVIRONMENTS

CODEMIDAS identifies implemented functionality, removes it to form a development task, and constructs behavioral tests grounded in the original implementation (Ye et al., 2026). The verifier is injected only at grading time. Related pipelines draw on issues and pull requests (Badertdinov et al., 2026; Chen et al., 2026a), commits (Jain et al., 2025), tests (Yang et al., 2025; Zhang et al., 2025), or documentation (Jain et al., 2024; Chen et al., 2026b). Their construction and filtering choices differ; our recovery experiments concern the specific final screen in CODEMIDAS.

2.2 GROUP-RELATIVE AND CRITIC-BASED OBJECTIVES

GRPO (Shao et al., 2024) replaces a learned value baseline with within-prompt reward statistics. Its variants address issues including clipping and train–inference mismatch (Yu et al., 2025; Ling Team, 2025). Critic-based approaches include VAPO for long reasoning chains (Yue et al., 2025) and SAO for asynchronous agentic RL (Hou et al., 2026). SAO combines single-rollout sampling, a double-sided importance-ratio mask, more frequent critic updates, frozen critic attention, and skip-observation GAE. It also identifies value pretraining as important. Earlier work, including CodeRL, uses execution outcomes to train a critic for code generation (Le et al., 2022). Our contribution is not the general idea of an execution-supervised critic, but its combination with recovered repository environments and construction-time validation records.

2.3 LONG-HORIZON ASYNCHRONOUS CODING

CODEMIDAS permits up to 500 interaction turns and 516,096 response tokens per rollout. Group-relative updates require all rewards in a group, so a long-running sibling delays the availability of that group's update. The original training system already allows maximum staleness 8; ASSAY changes the unit of consumption from a completed group to individual completed rollouts. This reduces a synchronization constraint, but does not eliminate off-policy error or guarantee lower end-to-end cost.

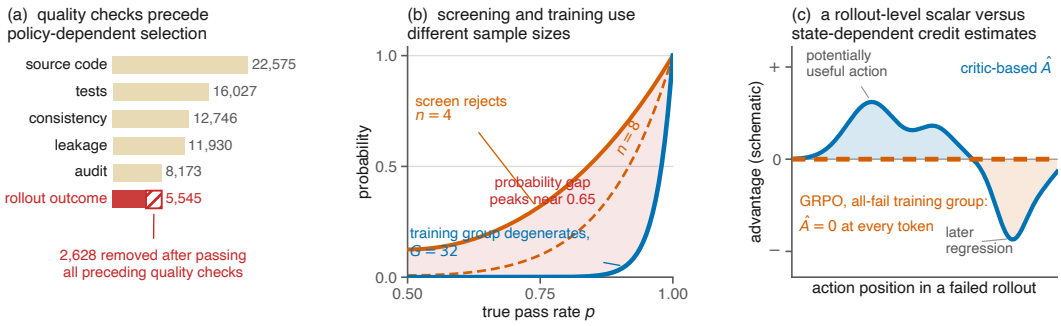


Figure 1: **Selection, degeneracy, and value estimation. (a)** The final rollout-outcome screen removes 2,628 environments after the preceding quality checks. **(b)** Closed-form unanimity probabilities for screening and training groups under a common success probability p. **(c)** Conceptual contrast between zero GRPO reward advantages in an all-fail training group and potentially state-dependent critic-based estimates. The curve is a mechanism schematic, not a measured trajectory or a guarantee of correct credit assignment; the recorded value trace appears in Appendix F.

3 Screening Outcomes and Training Utility

3.1 What a degenerate group establishes

Call a group *degenerate* when all of its verifier rewards agree.

Proposition 1. *If $r_1, \ldots, r_G \in \{0, 1\}$ are degenerate, then the group-relative advantages are zero, with or without standard-deviation normalization. Consequently, this group's reward-weighted policy-gradient contribution is zero.*

Proof. The group mean satisfies $\bar{r} = r_i$ for every rollout, so $r_i - \bar{r} = 0$. Standard-deviation normalization gives the same result with the usual ϵ-guarded denominator. Each reward-weighted gradient term therefore vanishes. □

The proposition concerns a sampled training group, not the quality or permanent learnability of a task. Separate regularization terms can still contribute gradients. Likewise, model updates from other tasks can change behavior on a task whose own sampled groups are degenerate. Passing execution and audit checks establishes the tested properties, not complete verifier correctness.

3.2 Finite-sample screening is not a training-degeneracy test

For conditionally independent attempts, let p_{scr} be a task's success probability under the screening model and p_t its probability under the training policy at step t. Then

$$q_{\text{rej}} = p_{\text{scr}}^n + (1 - p_{\text{scr}})^n, \qquad q_{\text{deg},t} = p_t^G + (1 - p_t)^G. \tag{1}$$

The two quantities differ in both policy and sample size. Figure 1b first holds the policy fixed, $p_{\text{scr}} = p_t = p$, to isolate the sample-size effect. At $p = 0.8$, $n = 4$ rejects with probability 0.4112, whereas $G = 32$ degenerates with probability 0.000792. At $p = 0.9$, the probabilities are 0.6562 and 0.0343. For $n = 4$ and $G = 32$, their difference reaches approximately 0.65 near $p = 0.93$. This difference is a gap between marginal event probabilities, not an empirical false-rejection rate or the probability of a joint event.

Increasing n reduces the same-policy gap while $n < G$, and closes it at $n = G$. It does not remove policy mismatch. In particular, four screening failures neither establish $p_t = 0$ nor imply that 32 training attempts will all fail. This is why the effect of pool recovery must be evaluated under each learner rather than inferred from screening labels.

3.3 Degeneracy changes during training

For $G > 1$, $q_{\text{deg}}(p, G)$ decreases as p approaches 0.5 and increases as p moves from 0.5 toward 1. An improvement from $p = 0.85$ to 0.95 therefore raises degeneracy at $G = 32$ from approximately 0.0055 to 0.1937, whereas improvement on a very difficult task can reduce it. Aggregate degeneracy

need not increase monotonically. A static screen cannot adapt to either change. We measure the fraction ρ_t of degenerate groups during training (§5.7); $1 - \rho_t$ is the fraction of groups eligible for a nonzero reward-driven update, not an exact measure of statistical effective batch size or GPU utilization.

3.4 What a state-dependent critic can provide

Let $\tau = [a_0, o_0, \ldots, a_M]$ alternate model actions and observations, with terminal reward $R(\tau) \in \{0, 1\}$. GRPO assigns the same reward-derived scalar to the generated tokens of one rollout. With a critic, SAO's skip-observation recursion bridges the last token of action a_i to the first token of the next action:

$$\hat{A}(a_{i,N}) = \delta_i + \gamma\lambda\hat{A}(a_{i+1,0}), \qquad \delta_i = r_i + \gamma V_\phi(s_{i+1,0}) - V_\phi(s_{i,N}). \tag{2}$$

Here $s_{i,j}$ is the causal prefix before the indexed generated token, r_i is zero except at termination, and terminal continuation value is zero. Within action spans, the usual token-level recursion applies. Observations remain part of the state context; they are skipped as policy-generated steps, not removed from the critic's input.

State dependence allows different advantage estimates within a failed trajectory, but does not guarantee that they have the correct signs. With $\gamma = \lambda = 1$, GAE reduces to $R(\tau) - V_\phi(s)$ (Schulman et al., 2016); a failed episode does not automatically provide positive credit for an intermediate edit. Similarly, undiscounted terminal success alone does not prefer shorter successful paths. VGVP supplies outcome-labeled states, including successful reference and agent records, from which value information can transfer. This is an initialization mechanism, not additional dense reward or a guarantee of learning from tasks with zero achievable success probability.

3.5 Observation-heavy trajectories

Observations account for 71.3% of tokens in the coding trajectories analyzed here, compared with 12.8% in the TIR comparison. Long source files and test logs therefore make the treatment of action–observation boundaries especially consequential. We use SAO's skip-observation estimator and evaluate its contribution through an ablation, rather than assuming that observation content is intrinsically uninformative.

4 Assay

4.1 Recovering tasks after the quality checks

We retain the first five stages of the CodeMidas funnel and bypass its final rollout-outcome screen (Table 10). The resulting RL pool is $\mathcal{D}_{\text{all}} = \mathcal{D}_{\text{keep}} \cup \mathcal{D}_{\text{disc}}$, with $5{,}545$ retained and $2{,}628$ recovered tasks. The recovered pool contains $1{,}704$ screening-all-fail and 924 screening-all-pass tasks. These names refer only to the original screen. Reference-patch sizes and screening lengths differ substantially between the subsets (Appendix E), motivating separate subset experiments.

Recovery preserves the earlier checks but does not certify the recovered environments. We add an online leakage guard (§4.4) and report its trigger rates. Neither this guard nor a low predicted value repairs a weak verifier or an underspecified task; these remain limitations of the recovered pool.

4.2 Verifier-grounded value pretraining

VGVP reuses labeled records from five construction stages: reference solutions, execution-consistency checks, solution-audit rollouts, adversarial leakage rollouts, and rollout-outcome screening. The full corpus contains $138{,}941$ records, of which $44{,}676$ (32.15%) come from recovered tasks (Table 6). These records are not all independent full agent trajectories: repeated consistency checks and reference-solution records must be distinguished from sampled solver paths. Only available action-boundary states contribute to the regression objective.

For a labeled record τ, let $\mathcal{S}(\tau)$ contain its available causal action-boundary states. We minimize

$$\mathcal{L}_{\text{VGVP}}(\phi) = \mathbb{E}_\tau \mathbb{E}_{s \in \mathcal{S}(\tau)} \left[\left(V_\phi(s) - \gamma^{d(s,\tau)} R(\tau) \right)^2 \right], \tag{3}$$

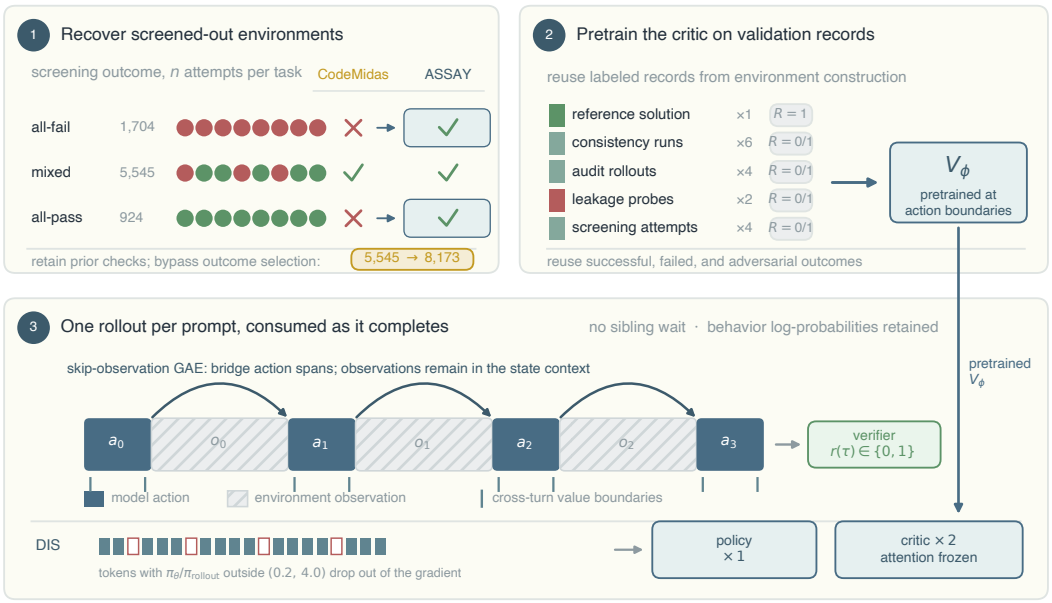


Figure 2: **The ASSAY training recipe.** The construction pipeline and its preceding quality checks are retained. The final screen's rejected tasks are recovered; construction-time outcome records initialize a critic; and completed single rollouts feed asynchronous actor–critic updates. Tick marks indicate cross-turn value-estimation boundaries. Observations are retained in the causal state, while the advantage recursion skips observation tokens.

where $d(s, \tau)$ is the remaining horizon in the chosen segmentation. We use $\gamma = 1$, so the target is the record's terminal label at every sampled state. Boundary sampling emphasizes the cross-turn transitions in Equation 2; it does not imply that the RL critic is queried only at boundaries.

Successful reference and agent records complement failed attempts. Unanimity at the task level is not evidence that individual labels are more reliable than those of mixed-outcome tasks: both have verifier-defined outcomes. Adversarial traces confirmed to recover the implementation through leakage receive $R = 0$, even if the verifier returned pass. This supplies negative supervision for shortcuts without treating verifier success as semantic correctness in every case. We pretrain for 12,000 steps and continue critic updates during RL. Because the corpus mixes construction policies, VGVP is an initialization rather than an unbiased estimate of the current policy's value function.

4.3 OPTIMIZATION

We follow SAO (Hou et al., 2026). Rollout generation records behavior-policy log-probabilities, giving the tokenwise ratio $w_t(\theta) = \exp(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\text{rollout}}(a_t \mid s_t))$. Let $f(w) = w\,\mathbf{1}\{1 - \epsilon_\ell < w < 1 + \epsilon_h\}$. The policy maximizes

$$J(\theta) = \hat{\mathbb{E}}_t \left[\text{sg}\Big(f(w_t(\theta)) \hat{A}_t \Big) \log \pi_\theta(a_t \mid s_t) \right], \tag{4}$$

where sg denotes stop-gradient. This makes the coefficient an importance-weighted, hard-masked policy-gradient weight; differentiating through it would define a different update even inside the accepted interval. We use $(\epsilon_\ell, \epsilon_h) = (0.8, 3.0)$, two critic updates per policy update, and trainable critic MoE projections with attention frozen. Completed rollouts accumulate into the training batch without waiting for prompt siblings. The mask limits extreme token ratios; it is not an exact correction for arbitrary trajectory staleness.

4.4 ONLINE LEAKAGE GUARD

A rule filter identifies suspect tool calls and an LLM judge assesses whether they attempt to obtain the intended implementation without doing the development work. On a confirmed hit, the call returns an uninformative result and the rollout continues. This avoids introducing an artificial terminal event. The design follows the coding-RL guard described in GLM-5 Team (2026); full rules and judge prompts appear in Appendix G. The guard targets attempted leakage, not silent false positives from

Policy	Training	SWE-bench Pro	DeepSWE v1.1	Program-Bench	RepoZero C2Rust	Terminal-B. v2.1	CODEMIDAS Val
MiMo-V2.5	none (initial)	50.3	10.0	4.5	40.5	63.7	35.0
	GRPO on $\mathcal{D}_{\text{keep}}$	54.4	21.7	21.5	51.8	72.2	44.7
	GRPO on $\mathcal{D}_{\text{all}}$	54.1	21.2	21.0	51.4	71.8	44.3
	ASSAY on $\mathcal{D}_{\text{keep}}$	56.1	24.3	24.8	54.2	74.6	47.1
	ASSAY on $\mathcal{D}_{\text{all}}$	57.9	27.5	28.1	56.8	76.9	49.8
GLM-5.3-Flash	none (initial)	58.6	63.4†	41.2	66.3	84.3†	57.4
	GRPO on $\mathcal{D}_{\text{keep}}$	60.1	65.2	44.8	68.5	85.6	60.2
	ASSAY on $\mathcal{D}_{\text{keep}}$	61.0	66.4	46.7	69.8	86.3	61.8
	ASSAY on $\mathcal{D}_{\text{all}}$	62.2	68.0	48.9	71.2	87.1	63.5

Table 1: **Main results.** MiMo-V2.5 initial and GRPO/$\mathcal{D}_{\text{keep}}$ scores are from Ye et al. (2026); the new runs use the corresponding evaluation settings. †Vendor-reported initial scores (Z.ai, 2026), retained as reference values rather than separate harness-matched measurements.

incomplete tests. Its trigger rate is therefore a diagnostic of suspected exploit behavior, not a direct estimate of verifier error.

5 EXPERIMENTS

5.1 SETUP

We train MiMo-V2.5 (Xiaomi MiMo Team, 2026), the policy used in Ye et al. (2026), and GLM-5.3-Flash (Z.ai, 2026), a stronger agentic-coding policy. Evaluation covers SWE-bench Pro (Deng et al., 2026), DeepSWE v1.1 (Huang & Jiang, 2026), ProgramBench (Yang et al., 2026), RepoZero C2Rust (Zhang et al., 2026), Terminal-Bench v2.1 (Merrill et al., 2026), and CODEMIDAS Val (200 held-out tasks, three attempts per task). ProgramBench reports Almost Solved, the fraction of tasks passing at least 95% of tests; the other metrics are pass rates. The expanded training pool is task-disjoint from these evaluation sets.

The standard MiMo comparisons use 70 policy-update steps and 1,024 rollouts per step: 32×32 for GRPO and $1{,}024 \times 1$ for ASSAY. Pool labels denote RL task eligibility. Cross-optimizer comparisons evaluate training recipes, not identical supervision: ASSAY additionally uses construction-time records for critic pretraining. We separately examine prompt exposure and resource usage. Configuration details are in Appendix B; reported scores are point estimates, without claims of statistical significance for small differences.

5.2 MAIN RESULTS

On $\mathcal{D}_{\text{keep}}$, ASSAY exceeds the MiMo GRPO baseline on all six metrics, including $+1.7$ points on SWE-bench Pro and $+2.6$ on DeepSWE. With $\mathcal{D}_{\text{all}}$, the scores reach 57.9 and 27.5, respectively. Separating these two comparisons avoids attributing the full $+3.5$ and $+5.8$ improvements to the optimizer alone. For GLM-5.3-Flash, ASSAY/$\mathcal{D}_{\text{keep}}$ also improves all six reported scores relative to GRPO/$\mathcal{D}_{\text{keep}}$, and pool expansion raises SWE-bench Pro from 61.0 to 62.2 and DeepSWE from 66.4 to 68.0. The stronger policy therefore still benefits from the recipe. The expanded-pool GRPO comparison is available for MiMo only.

5.3 THE EFFECT OF RECOVERING SCREENED-OUT TASKS

The transition from $\mathcal{D}_{\text{keep}}$ to $\mathcal{D}_{\text{all}}$ improves ASSAY by 1.8, 3.2, and 2.7 points on SWE-bench Pro, DeepSWE, and CODEMIDAS Val. Under GRPO the corresponding changes are -0.3, -0.5, and -0.4. Thus, expanding this pool helps one evaluated recipe but not the other; the small GRPO differences do not prove that every recovered task is useless or that no alternative GRPO configuration can benefit.

Adding only the 1,704 screening-all-fail tasks yields $57.1/26.3/48.9$ on the same three metrics; adding only the 924 screening-all-pass tasks yields $56.4/24.9/47.6$. Both are above ASSAY/$\mathcal{D}_{\text{keep}}$, with a larger gain from the all-fail subset. Because these subsets differ in size, difficulty, and composition, the contrast does not isolate a per-task effect of screening outcome.

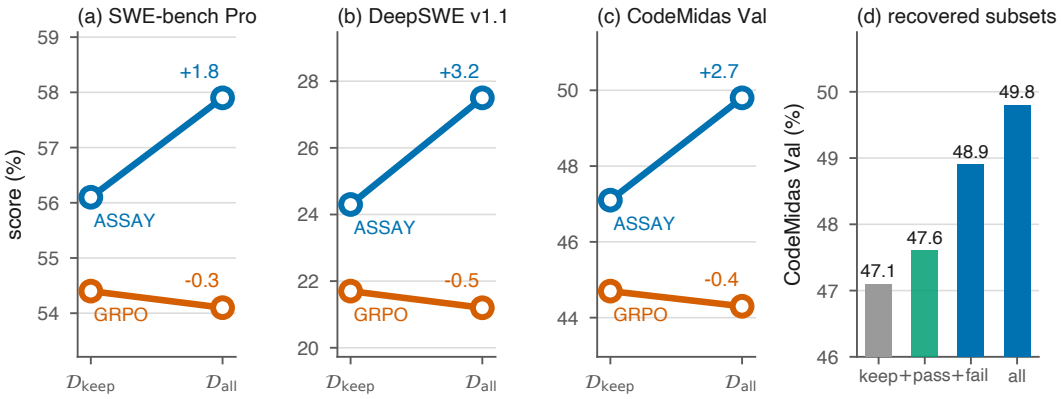


Figure 3: **Pool expansion under two training recipes, MiMo-V2.5. (a–c)** Adding the same recovered task pool improves ASSAY and produces small decreases under the evaluated GRPO configuration. **(d)** Adding the screening-all-fail subset yields a larger gain than adding the screening-all-pass subset. Screening labels do not establish future training-group degeneracy.

Configuration	SWE-bench Pro	DeepSWE v1.1	CODEMIDAS Val
ASSAY (full, $\mathcal{D}_{\text{all}}$)	57.9	27.5	49.8
w/o verifier-grounded pretraining	55.9	24.1	46.6
w/o pool recovery ($\mathcal{D}_{\text{keep}}$)	56.1	24.3	47.1
w/o skip-observation GAE	56.3	25.0	47.4
w/o frozen-attention critic	55.2	23.4	45.8
w/o faster value update ($K = 1$)	56.6	25.4	47.9
w/o DIS mask	52.8	19.6	41.2
w/o online leakage guard	57.1	26.4	48.5

Table 2: **Leave-one-component-out ablations, MiMo-V2.5.** Except for the pool-recovery ablation, RL uses $\mathcal{D}_{\text{all}}$. These comparisons estimate changes from the full recipe; they do not identify a factorial interaction between VGVP and pool recovery.

5.4 COMPONENT ABLATIONS

Removing VGVP reduces CODEMIDAS Val from 49.8 to 46.6; removing pool recovery reduces it to 47.1. Skip-observation GAE, attention freezing, and the faster critic update each contribute positive margins in this configuration. The largest drop is without DIS, to 41.2, consistent with the importance of off-policy stabilization. The guard ablation reaches 48.5, below the full recipe. These are conditional component effects, not proof that the components are individually necessary. Pretraining-length and learning-rate sensitivity results appear in Table 4.

5.5 CONTROLLING PROMPT EXPOSURE

SAO's original comparison uses 128×1 versus 16×8 sampling (Hou et al., 2026). Our standard configuration similarly gives ASSAY 32 times more prompt slots per update than GRPO. Table 11 examines this difference. Matching 1,024 unique prompts per batch while retaining $G = 32$ raises GRPO's CODEMIDAS Val score from 44.7 to 46.2, compared with ASSAY's 47.1, but uses 32,768 rather than 1,024 rollouts per update. At the standard rollout count, reducing GRPO to $G = 4$ yields 45.3. Prompt exposure therefore explains a substantial part of the standard comparison. These controls do not simultaneously match prompt count, rollout count, extra supervision, and wall-clock cost.

5.6 TRAINING COST

Figure 8 reports the resource measurements. Relative to GRPO, ASSAY increases peak memory from 61 to 78 GB per device and training compute from 18 to 31 GPU-hours per step, while rollout GPU-hours decrease from 412 to 268. The measured within-group slowest-to-median rollout-time ratio is 4.3 at $G = 32$. Wall-clock time for the reported 70-step run decreases from 196 to 142 hours. These results are consistent with reduced group waiting offsetting the critic's overhead in this setup,

not with a hardware-independent speedup guarantee. Standalone VGVP and judge costs are not separately itemized, so the figures should not be read as a complete equal-budget comparison of end-to-end training.

5.7 Training dynamics

The measured GRPO degenerate-group fraction rises from approximately 8% to 31% over the run (Figure 6a). This is an observation about these policies and sampled tasks, not a necessary consequence of improving mean pass rate. VGVP raises initial explained variance from 0.05 to 0.42; at step 70 the cold critic reaches 0.71, versus approximately 0.78 with VGVP. The largest separation is early, supporting better initialization without showing identical asymptotes. The DIS mask rate approaches 5.8%. A nonzero mask rate describes how often ratios leave the accepted interval; it is not by itself a stability certificate.

5.8 Behavioral diagnostics

We use Ye et al. (2026)'s definitions of exploration, drafting, and self-verification (Appendix H). Under ASSAY, pre-edit read/search calls increase from 27.2 to 44.8, drafting ratio from 0.358 to 0.671, and post-edit verification commands from 2.03 to 2.79. We also track recovered tasks with no observed successful rollout in the analyzed samples (Figure 11). Their behavior changes despite unchanged observed success. This diagnostic is compatible with both critic-mediated updates and cross-task transfer; it does not establish zero true success probability or uniquely identify interior credit assignment.

6 Limitations

Scope and controls. Recovery is tested on one construction pipeline and two policy families. The full expanded-pool GRPO comparison is limited to MiMo. The reported component ablations do not establish a VGVP–recovery interaction, and a matched decomposition of RL-pool membership and critic-pretraining-corpus membership remains necessary for finer attribution. Single-rollout baselines without a learned critic and alternative GRPO sampling strategies could further test the mechanism.

Value supervision. Repeated checks are correlated records rather than independent demonstrations. Terminal labels and overall explained variance do not certify intermediate action values; branching continuations from fixed states would provide a stronger calibration test. Construction-policy shift and asynchronous state-distribution shift remain sources of bias.

Reliability and uncertainty. The guard cannot detect every exploit or a weak test that silently accepts incorrect code (Liu et al., 2023; Wang et al., 2026). Our scores lack seed-level confidence intervals, and prompt-level dependence matters for the 200-task validation set. Hardware-normalized, fully itemized end-to-end cost is also needed before generalizing the measured resource advantage.

7 Conclusion

Rollout-outcome filtering mixes environment reliability with policy-dependent training utility. ASSAY retains existing quality checks, recovers $2{,}628$ screened-out tasks, and combines construction-grounded value initialization with asynchronous actor–critic training. Recovery improves the evaluated ASSAY recipe while leaving the MiMo GRPO comparison approximately unchanged. Pipelines should therefore retain intermediate environments and their outcome records, and evaluate task eligibility with the learner that consumes them rather than treat one offline screen as a permanent verdict on data value.

AI USE STATEMENT

During manuscript preparation, generative AI tools assisted with drafting, grammar check and polishing. They also assisted with LaTeX and document-processing code, figure-label updates and layout checks. AI-generated suggestions are not treated as independent scientific evidence. The authors retain responsibility for independently validating AI-assisted text, code, figures, citations and scientific claims, and for the final content of the submission.

ETHICS STATEMENT

This work studies coding agents in executable software environments. Its main ethical and safety concerns include repository provenance and licensing, possible exposure of secrets or personal information in code and execution traces, reward exploitation, and misuse of more capable autonomous coding agents. The retained construction checks and online leakage guard (§4.1, §4.4) target specific reliability and leakage failures; neither verifier success nor the guard establishes general correctness, security, or safety. Reuse or redistribution of repository snapshots, reference implementations, and agent traces must respect source licenses, benchmark terms, and access restrictions, and should exclude sensitive content. Executing agent-generated code calls for isolated environments, restricted credentials and network access, and human review before deployment. These are requirements for responsible reuse and deployment, rather than guarantees supplied by the reported benchmark performance.

REPRODUCIBILITY STATEMENT

Task eligibility and pool composition are described in §4.1 and Appendix E; the value-pretraining objective and record inventory are given in §4.2 and Appendix D. Algorithm 1 and Appendix B specify the update sequence and training configuration, while §5.1, Appendix G, and Appendix H document evaluation metrics, prompt templates, and behavioral measures. Section 3 and Appendix C state the assumptions and derivations for the analytical results. The accompanying manuscript-source package contains numerical figure inputs, figure-generation scripts, dependencies, and build and consistency checks. It reproduces the figures from the reported inputs, not the model-training runs. Appendix J identifies the additional run-level artifacts and metadata needed for independent experimental replication and the resulting limits on reproducibility.

REFERENCES

Ibragim Badertdinov, Maria Nekrashevich, Alexander Shevtsov, and Anton Golubev. SWE-rebench v2: Language-agnostic SWE task collection at scale. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=UCAda9kS57.

Mouxiang Chen, Lei Zhang, Yifan Feng, Xin Wang, Wei Zhao, Ruisheng Cao, Jian Yang, Jiajun Chen, Ming Li, et al. SWE-Universe: Scale real-world verifiable environments to millions. *arXiv preprint arXiv:2602.02361*, 2026a. URL https://arxiv.org/abs/2602.02361.

Yuxuan Chen, Shuo Chang, Karan Chawa, Fei Lin, Bo Chen, Song Wang, and Ahmed E. Hassan. MindForge: Teaching small language models whole-life-cycle software engineering via source-free program synthesis. *arXiv preprint arXiv:2607.27146*, 2026b. URL https://arxiv.org/abs/2607.27146.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chris Rane, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=uEVTdoAbnK.

Dian Fu, Shuai Wu, Yifan Wu, Zhen Peng, Yan Huang, Jie Sun, Jian Zeng, Ming Jiang, Lei Zhang, Yang Li, et al. daVinci-Env: Open SWE environment synthesis at scale. *arXiv preprint arXiv:2603.13023*, 2026. URL https://arxiv.org/abs/2603.13023.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026. URL https://arxiv.org/abs/2607.07508.

William Huang and Peter Jiang. DeepSWE v1.1, 2026. URL https://deepswe.datacurve.ai/blog/deepswe-v1-1.

Naman Jain, Manish Shetty, Tianjun Zhang, King Han, Koushik Sen, and Ion Stoica. R2E: Turning any GitHub repository into a programming agent environment. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 21196–21224, 2024. URL https://proceedings.mlr.press/v235/jain24c.html.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Lianmin Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022.

Ling Team. Every step counts: Decoupling trajectories for stable agentic reinforcement learning. *arXiv preprint arXiv:2510.18855*, 2025. URL https://arxiv.org/abs/2510.18855. IcePop.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572, 2023.

Mike A. Merrill, Alex Gu Shaw, Nicholas Carlini, Bowen Li, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

You Wang, Michael Pradel, and Zhuo Liu. Are "solved issues" in SWE-bench really solved correctly? an empirical study. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 169–181, 2026.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems*, volume 38, pp. 78500–78525, 2025.

Xiaomi MiMo Team. MiMo-V2.5, 2026. URL https://mimo.xiaomi.com/mimo-v2-5.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

John Yang, Kilian Lieret, Jiayi Ma, P. Thakkar, Dmitry Pedchenko, Sander Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. ProgramBench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026. URL https://arxiv.org/abs/2605.03546.

Bowen Ye, Lei Li, Shicheng Li, Zihao Yue, Linghao Zhang, Hanglong Lv, Yuanxin Liu, Wenhan Ma, Hao Tian, Rang Li, Jinhao Dong, Yikai Zhao, Xiangwei Deng, Hailin Zhang, Liang Zhao, Qi Liu, Lingpeng Kong, Tong Yang, and Fuli Luo. CodeMidas: Scaling agentic coding RL environments from code itself. *arXiv preprint arXiv:2609.22068*, 2026. URL https://arxiv.org/abs/2609.22068.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, Tiantian Fan, Zhengyin Du, et al. VAPO: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025. URL https://arxiv.org/abs/2504.05118.

Z.ai. GLM-5.3-Flash, 2026. URL https://huggingface.co/zai-org/GLM-5.3-Flash. 320B-A18B mixture-of-experts, 1M context, MIT license.

Yao Zeng, Shuai Li, Di Dong, Rui Xu, Zhi Chen, Lin Zheng, Yang Li, Zhen Zhou, Hao Zhao, Lei Tian, et al. SWE-Hub: A unified production system for scalable, executable software engineering tasks. *arXiv preprint arXiv:2603.00575*, 2026. URL https://arxiv.org/abs/2603.00575.

Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zhongxin Cui, Binyuan Hui, and Junyang Lin. Synthesizing software engineering data in a test-driven manner. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 76518–76540, 2025. URL https://proceedings.mlr.press/v267/zhang25cn.html.

Zhenhao Zhang, Yang Xu, Jiacheng Liang, Wei Li, Xin Chen, Long Qian, Xiaoyu Pei, Jie Huang, Rui Sun, and Yong Wu. RepoZero: Can LLMs generate a code repository from scratch? *arXiv preprint arXiv:2605.07122*, 2026. URL https://arxiv.org/abs/2605.07122.

A ALGORITHM

Algorithm 1 separates task selection and value initialization from SAO's asynchronous optimizer. One rollout per prompt does not mean one rollout per policy update: completed trajectories still accumulate into a training batch.

Algorithm 1 ASSAY: asynchronous actor–critic training on recovered environments

Input: RL pool $\mathcal{D}_{\text{all}} = \mathcal{D}_{\text{keep}} \cup \mathcal{D}_{\text{disc}}$; policy π_θ; critic V_ϕ; batch size $B = 1{,}024$; trust region $(\epsilon_\ell, \epsilon_h)$; $K = 2$
$\mathcal{T} \leftarrow$ construction-time records with outcome labels and available causal states
initialize V_ϕ from the policy and optimize $\mathcal{L}_{\text{VGVP}}(\phi; \mathcal{T})$ for 12,000 steps
freeze critic attention; leave MoE projections trainable
launch workers, each sampling one task and one trajectory under its behavior policy, recording token log-probabilities and applying the leakage guard
while training **do**
    collect B completed trajectories; grade each with the hidden verifier
    compute token-level GAE, using Equation 2 at cross-turn boundaries and zero terminal continuation value
    compute behavior-to-current-policy ratios $w_t(\theta)$
    take one ascent step on Equation 4, with advantage weights detached
    take K critic regression steps on the batch's return targets
    publish updated policy weights to the rollout workers
end while

B TRAINING AND EVALUATION CONFIGURATION

Setting	CODEMIDAS GRPO (Ye et al., 2026)	ASSAY
Algorithm	GRPO	SAO (single-rollout, async)
Reward	binary verifier	binary verifier
Advantage normalization	std disabled	skip-observation GAE
Batch size	32 prompts	1,024 prompts
Rollouts per task	32	1
Value model	none	policy init. + VGVP (App. D)
Critic updates / policy update	—	2
Critic trainable modules	—	MoE projections (attention frozen)
$(\epsilon_\ell, \epsilon_h)$	—	$(0.8,\ 3.0)$
γ, λ_{policy}, λ_{critic}	—	1.0, $1 - 1/(\alpha l)$ (α=1.5), 1.0
Max prompt / response tokens	8,192 / 516,096	8,192 / 516,096
Max turns per rollout	500	500
Maximum staleness	8	no explicit cap; DIS masking
Optimizer	Adam $(0.95, 0.95)$, ϵ=10^{-15}	same
Policy / critic LR	5×10^{-6} / —	5×10^{-6} / 1×10^{-5}
Warmup / grad clip / WD	0 / 1 / 0	10 (critic) / 1 / 0

Table 3: Training configurations. The GRPO column follows Ye et al. (2026), Table A1. The table specifies the settings used here rather than claiming that every hyperparameter is unchanged from SAO.

Sensitivity. Table 4 varies four settings independently. The coding trust region outperforms the TIR setting on these metrics. Increasing critic updates from two to four provides no further gain. Training attention at one tenth the learning rate narrows the gap to freezing it, leaving learning-rate sensitivity as a competing explanation for part of the freezing benefit. Increasing VGVP from 12k to 24k steps changes each score by 0.1 point; no significance claim is made for this difference.

C THE ARITHMETIC OF DEGENERATE GROUPS

The following probabilities assume conditionally independent Bernoulli attempts. They are distinct from measured training-group frequencies.

Setting	Value	SWE-bench Pro	DeepSWE v1.1	CODEMIDAS Val
Trust region $(\epsilon_\ell, \epsilon_h)$	$(0.3,\ 5.0)$ *SAO's TIR*	56.8	25.9	48.2
	$(0.8,\ 3.0)$ *SAO's coding*	57.9	27.5	49.8
	$(0.8,\ 1.0)$	55.4	23.8	46.1
	no mask	52.8	19.6	41.2
Critic updates K	1	56.6	25.4	47.9
	2	57.9	27.5	49.8
	4	57.6	27.0	49.3
Critic attention	frozen	57.9	27.5	49.8
	trained, lr $\times 0.1$	57.1	26.3	48.6
	trained, full lr	55.2	23.4	45.8
VGVP pretraining steps	0 (cold)	55.9	24.1	46.6
	4k	56.9	26.0	48.3
	12k	57.9	27.5	49.8
	24k	58.0	27.6	49.9

Table 4: Sensitivity sweeps, MiMo-V2.5 on $\mathcal{D}_{\text{all}}$. The full recipe uses $(0.8, 3.0)$, $K = 2$, frozen attention, and 12k VGVP steps. These comparisons do not establish a unique mechanism for freezing or pretraining.

Screening and training are different events. For a fixed policy with task success probability p, a screen of n attempts is unanimous with probability $p^n + (1-p)^n$, and a training group of size G is unanimous with probability $p^G + (1-p)^G$. Table 5 holds the policy fixed to isolate the sample-size effect. Its final column is a difference of marginal probabilities, not the probability that a rejected task would produce a nondegenerate training group. For independent screening and training samples, the latter joint probability is $q_{\text{rej}}(1 - q_{\text{deg}})$.

	screen rejects		group degenerates		
p	n=4	n=8	G=8	G=32	probability gap
0.50	0.125	0.008	0.008	0.000	+0.125
0.60	0.155	0.017	0.017	0.000	+0.155
0.70	0.248	0.058	0.058	0.000	+0.248
0.80	0.411	0.168	0.168	0.001	+0.410
0.85	0.523	0.272	0.272	0.006	+0.517
0.90	0.656	0.430	0.430	0.034	+0.622
0.95	0.815	0.663	0.663	0.194	+0.621
0.98	0.922	0.851	0.851	0.524	+0.398

Table 5: Closed-form probabilities $p^k + (1-p)^k$, rounded to three decimals. The final column subtracts the $G = 32$ probability from the $n = 4$ probability. The functions are symmetric about $p = 0.5$; displayed zeros may reflect rounding.

What a rollout budget buys. With budget $B = 1{,}024$ rollouts, group size G buys B/G prompt groups. A fraction $1 - p^G - (1-p)^G$ are nondegenerate, so the expected eligible groups per rollout are

$$U(G) = \frac{1}{G}\mathbb{E}_p[1 - p^G - (1-p)^G]. \tag{5}$$

For mean-centered GRPO, $U(1) = 0$: with an ϵ-guarded denominator the estimator is defined but gives zero reward-driven advantage. For binary rewards, $U(2) = U(3) = \mathbb{E}[p(1-p)]$; these maximize this counting statistic over integer $G \geq 2$. Under the example $\text{Beta}(1.4, 2.4)$ distribution, $1{,}024$ rollouts yield approximately 189 eligible groups at $G = 2$ and 31 at $G = 32$. The distribution is an analytical example, not a fit to the task pool.

Eligible-group count is not gradient quality or final performance. Larger groups change baseline variance, while actor–critic training introduces critic error and additional compute. It does not force advantages to zero because of sibling unanimity, but a single rollout can still have zero or uninformative advantage. Thus Figure 4 motivates sampling controls rather than identifies the performance-optimal group size.

Recovered-pool size. The final funnel transition is $8{,}173 \to 5{,}545$ (Table 10), giving $|\mathcal{D}_{\text{disc}}| = 2{,}628$. The original screening outcomes partition these tasks into $1{,}704$ all-fail and 924 all-pass tasks.

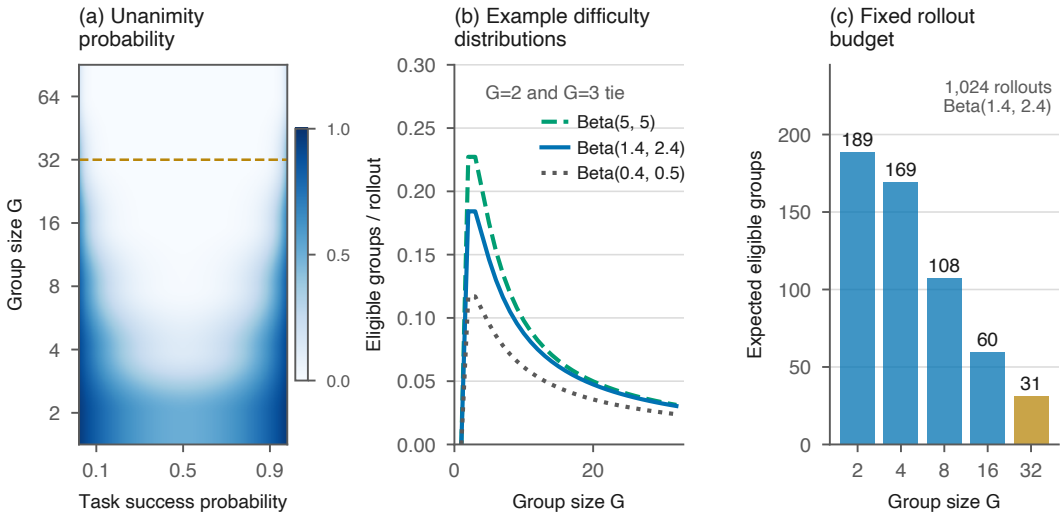


Figure 4: **Analytical sampling tradeoffs. (a)** Unanimity probability. **(b)** Eligible groups per rollout under three example difficulty distributions. **(c)** Expected eligible-group counts for 1,024 rollouts under $\mathrm{Beta}(1.4, 2.4)$. Gold marks $G = 32$. These are closed-form calculations, not observed counts or useful gradient magnitudes.

D VERIFIER-GROUNDED VALUE PRETRAINING

Source	Labeled records	From $\mathcal{D}_{\text{keep}}$	From $\mathcal{D}_{\text{disc}}$
Reference solutions	8,173	5,545	2,628
Execution-consistency runs	49,038	33,270	15,768
Solution-audit rollouts	32,692	22,180	10,512
Adversarial leakage rollouts	16,346	11,090	5,256
Rollout-outcome screening	32,692	22,180	10,512
Total	138,941	94,265	44,676

Table 6: VGVP record inventory. There are 17 records per task: one reference, six consistency checks, four solution audits, two adversarial attempts, and four screening attempts. Recovered tasks contribute 32.15% of the records. Repeated checks and reference records are not independent full solver trajectories.

Record granularity and targets. Equation 3 samples available causal states at action boundaries. With $\gamma = 1$, all states from one record receive the same terminal target. A verifier invocation or reference patch alone is not a full agent trajectory, and repeated execution does not create independent progress supervision. Reproducing VGVP requires state-bearing records, not merely the counts in Table 6. Confirmed leakage successes are relabeled $R = 0$.

Initialization versus online fitting. Figure 6 reports higher initial explained variance with VGVP, while both critics improve during RL. This supports an early initialization benefit. It does not imply that recovered tasks are visited only once, that online fitting cannot use them later, or that an accurate outcome predictor necessarily identifies correct intermediate actions.

E COMPOSITION OF THE RECOVERED POOL

The recovered subsets differ in reference-patch size, observed screening length, and language composition. These statistics describe selection effects; they cannot rule out residual task or verifier defects. All three subsets passed the preceding checks by construction.

Property	$\mathcal{D}_{\text{keep}}$ (5,545)	$\mathcal{D}_{\text{disc}}$ all-fail (1,704)	$\mathcal{D}_{\text{disc}}$ all-pass (924)	$\mathcal{D}_{\text{all}}$ (8,173)
Median reference solution (lines)	142	281	68	164
Interquartile range	66–305	131–548	29–122	71–342
Reference patch touches ≥ 2 files	65.9%	78.4%	41.2%	65.7%†
Median turns, screening rollouts	112	287	34	128
Distinct codebases	3,185	1,106	642	4,190
Execution-consistency pass	100%	100%	100%	100%
Leakage audit pass	100%	100%	100%	100%
Verifier-agreement audit pass	100%	100%	100%	100%

Table 7: Pool composition. All-fail and all-pass refer to screening outcomes, not true success probabilities or quality certificates. Published retained-pool statistics are from Ye et al. (2026). †The pooled multi-file percentage is recomputed from the displayed subgroup sizes and percentages, rounded to one decimal.

Language	$\mathcal{D}_{\text{keep}}$	all-fail	all-pass	Domain	$\mathcal{D}_{\text{keep}}$	all-fail	all-pass
Python	21.4%	17.2%	26.8%	Systems	17.4%	23.1%	9.6%
TypeScript	18.3%	16.4%	21.0%	Web	14.6%	11.8%	18.9%
Go	16.2%	17.9%	14.1%	Dev tools	13.6%	12.4%	16.2%
C++	12.5%	17.6%	6.8%	AI/ML	9.1%	10.6%	7.4%
JavaScript	11.3%	8.9%	14.7%	Data science	7.6%	6.2%	9.8%
C	7.4%	10.8%	4.1%	Multimedia	7.1%	8.4%	5.5%
Java	6.0%	5.1%	7.2%	Specialized	6.9%	9.1%	4.2%
Other (16)	6.9%	6.1%	5.3%	Other (9)	23.7%	18.4%	28.4%

Table 8: Language and domain shares. Systems and C/C++ are more frequent in the screening-all-fail subset; several scripting-language and web categories are more frequent in the screening-all-pass subset. Recovery changes composition as well as observed difficulty, and the subset comparison does not separate those factors.

F Case Study: Value Estimates Along a Failed Rollout

We examine a YAML-driven file-walker task whose `data_spider` entry point must omit dot-prefixed entries unless `hidden=True` is passed. This requirement also appears in the example in Ye et al. (2026). The trajectory analyzed here ends in failure. It provides a value diagnostic without assuming that the published training example was excluded by the original screen.

Turn	Agent action	$V_\phi(s_t)$	What changed
1–3	reads `README`, lists the package	0.06–0.09	initial exploration
4	opens the caller `from_yaml`	0.21	finds the real entry point
6	re-reads the statement, notes `hidden`	0.34	requirement now explicit
8	writes the `data_spider` skeleton	0.46	structure is right
11	adds `entry.startswith('.')` guard	0.62	adds conditional filtering
13	extends the guard to nested directories	0.44	over-application
15	one test now fails on a nested dotfile	0.26	symptom, not cause
17	deletes the guard to make that test pass	0.07	removes conditional filtering
18	submits	—	$r(\tau) = 0$

Table 9: Actions and predicted values along one failed rollout. Value changes are not independently measured continuation success rates or evidence that the corresponding policy advantages have the same signs.

Scope. The example shows that the learned value function is not constant within one failed trajectory. Testing calibration requires independent continuations from the same states. The behavioral measurements in Section 5.8 provide a broader descriptive complement, but neither analysis isolates the cause of policy improvement.

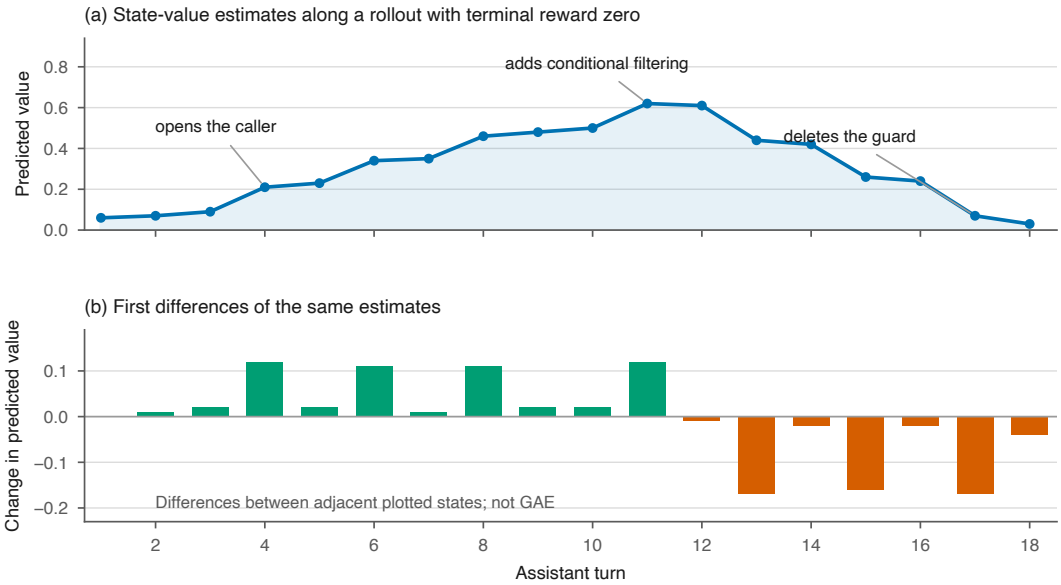


Figure 5: **Value diagnostics for a failed rollout.** **(a)** Predicted state values. **(b)** First differences between adjacent plotted turns. Panel (b) is a derived diagnostic, not GAE, an action advantage, or a direct measurement of policy-gradient credit.

G PROMPT TEMPLATES

ASSAY adds one agentic component to the pipeline, the online leakage guard, and one labelling rule for value pretraining. Both are given in full. Prompts belonging to CODEMIDAS's construction agents are unchanged and are documented in Ye et al. (2026).

Stage 1: rule filter (high recall, no model call). A tool call is forwarded to the judge if any pattern matches. The filter is deliberately over-inclusive; precision is the judge's job.

Leakage guard — rule filter

```
# path patterns reaching outside the intended working tree
  *.eval/*, *secret*, *golden*, *expected_output*, *.cache/*, *dist/*, *build/*
# reads of any file the construction agents are known to leave behind
  *.codemidas/*, *reference_solution*, *original_impl*
# any network egress
  curl, wget, pip download, git clone, npm install --registry
# search for the removed symbol outside the solver's files
  grep/rg for the target public symbol within site-packages, node_modules, vendor
# reads of any test file not present in the solver's starting state
```

Stage 2: judge (precision). The judge sees only the suspect call and its surrounding context, never the reference solution. This limits direct exposure of the intended implementation through the guard.

Leakage guard — intent judge

```
You are auditing a single tool call made by a coding agent that is solving a programming
task.  Decide whether this call is an attempt to obtain the intended implementation without
writing it, as opposed to legitimate exploration.

TASK STATEMENT:
{statement}

THE AGENT'S LAST THREE ACTIONS:
{recent_actions}

THE CALL UNDER REVIEW:
{tool_call}

Legitimate:  reading project source the agent is meant to modify; reading documentation;
running the agent's own tests; searching the codebase for callers or type definitions.
Not legitimate:  reading compiled or cached artefacts of the removed implementation;
retrieving the upstream project from the network; reading hidden verifier files; locating an
installed copy of the same package to copy from.

Answer with one line, VERDICT: legitimate or VERDICT: exploit, followed by one sentence of
justification.
```

Guard action. On `exploit`, the call returns a syntactically valid but uninformative result and the rollout continues. The intervention does not replace the remaining episode with an artificial terminal failure. Its effect on subsequent observations is part of the guarded training environment; it does not repair verifier coverage.

Value-pretraining labels. VGVP regresses on terminal outcomes, but the verifier's verdict is not always the outcome we want the critic to learn. The relabelling rule is:

VGVP label assignment

```
for each validation record t:
  if  t is a reference-solution replay              -> R = 1
  elif t is a starting-state consistency run      -> R = 0
  elif t is an adversarial leakage rollout:
    if the reviewing agent confirmed it        -> R = 0
       (even when the verifier returned pass)
    else                                  -> R = verifier verdict
  elif t is a solution-audit rollout              -> R = verifier verdict
  else (screening attempt)                       -> R = verifier verdict

states = available causal action-boundary states
target = gamma^d(s) * R,   where d(s) = remaining boundaries
```

The adversarial branch distinguishes verifier success from legitimate task completion. Relabeling a confirmed exploit supplies a negative value target; it does not guarantee detection of unseen shortcuts.

H Behavioral Metric Definitions

We reuse the definitions of Ye et al. (2026) verbatim so that our numbers are comparable to theirs. **Codebase exploration**: distinct read/search requests before the first codebase edit, deduplicating identical file/line-range reads and identical queries, and treating shell equivalents as duplicates. **Code drafting**: distinct 16-character fragments sampled at stride four from each `Write`/`Edit` payload, counted if present anywhere in preceding reasoning; the ratio is the summed count over fragments sampled. **Self-verification**: distinct verification commands after the final codebase edit, counting repeats once.

I SUPPORTING FIGURES AND TABLES

Stage	Property checked	Retained	Type
Task design	entry points and task scope	22,575	quality
Test construction	behavioral assertions	16,027	quality
Execution consistency	fail→pass in 6 containers	12,746	quality
Leakage filtering	attempted implementation recovery	11,930	quality
Solution audit	verdict–statement agreement	8,173	quality
Rollout-outcome filter	mixture of passes and failures	5,545	selection

Table 10: The CODEMIDAS funnel (Ye et al., 2026). The final stage selects by a screening policy's outcomes rather than directly auditing a particular defect. Earlier checks are not comprehensive quality proofs. ASSAY uses the 8,173-task pool before final selection.

Arm	Batch	Uniq. prompts	Rollouts	CODEMIDAS Val
A ASSAY	1024×1	1024	1024	47.1
B GRPO (published)	32×32	32	1024	44.7
C GRPO (matched prompts)	1024×32	1024	32768	46.2
D GRPO (G=4)	256×4	256	1024	45.3

Table 11: **Prompt-exposure controls on $\mathcal{D}_{\text{keep}}$.** Batch entries are prompts × rollouts per prompt. C matches A's prompt count using 32 times more rollouts per update. D matches the standard rollout count, not measured GPU cost. These controls do not match all resource and supervision dimensions simultaneously.

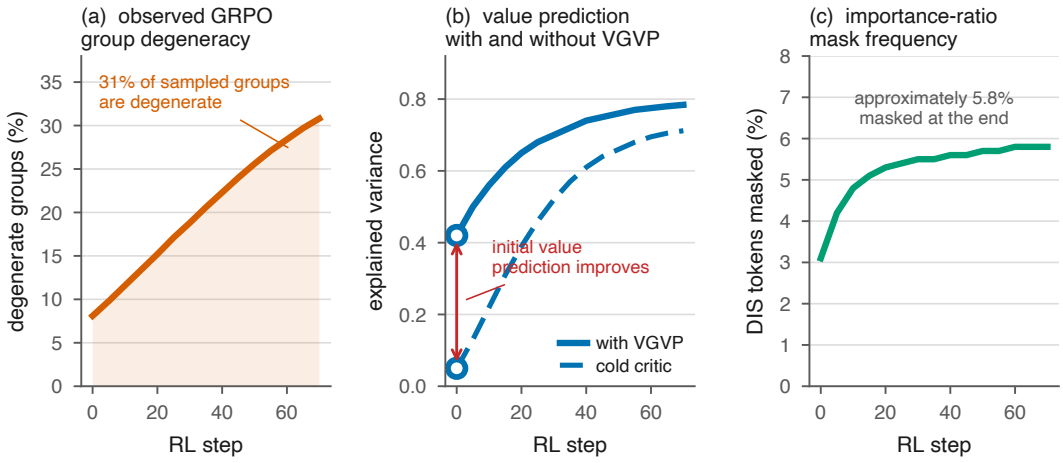


Figure 6: **Training diagnostics. (a)** Approximately 31% of sampled GRPO groups are degenerate at the end of the run; this is not the fraction of GPU time lost. **(b)** VGVP improves initial explained variance, and the gap narrows during training. **(c)** The fraction of masked tokens approaches 5.8%. These diagnostics alone do not identify a causal mechanism or certify stability.

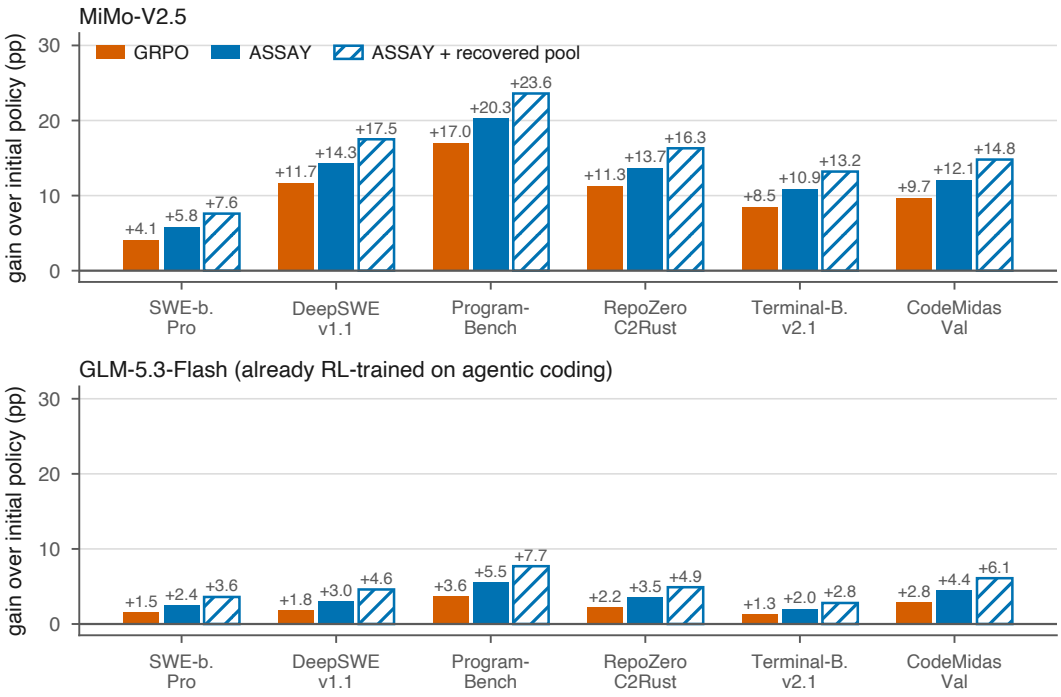


Figure 7: **Gains over the listed initial scores.** Values are derived from Table 1; percentage-point gains are not difficulty-normalized effects. GLM DeepSWE and Terminal-Bench reference scores are vendor-reported, as marked in that table.

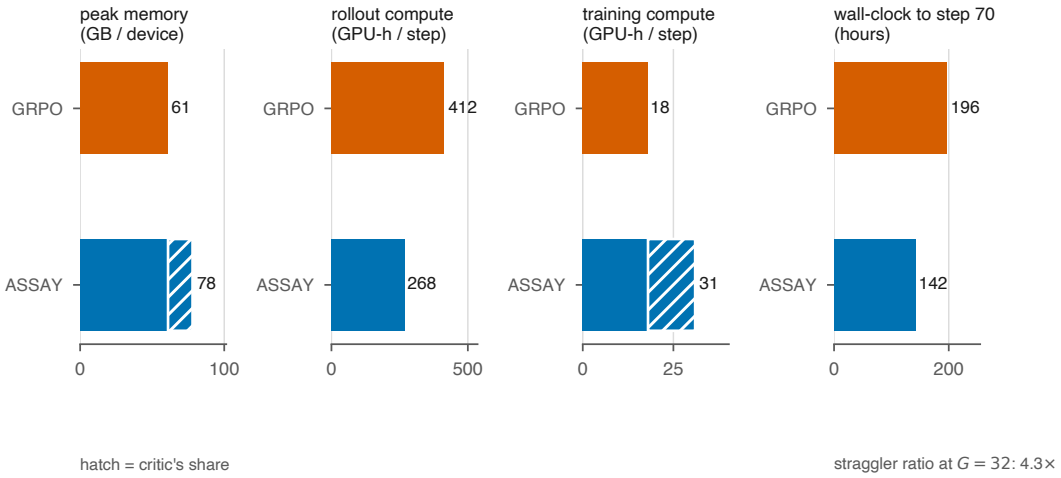


Figure 8: **Reported training-resource comparison.** The critic increases memory and update compute; the asynchronous recipe uses fewer rollout GPU-hours and less wall-clock time in this setup. The within-group slowest-to-median rollout-time ratio is 4.3 at $G = 32$. Standalone VGVP and judge costs are not separately itemized.

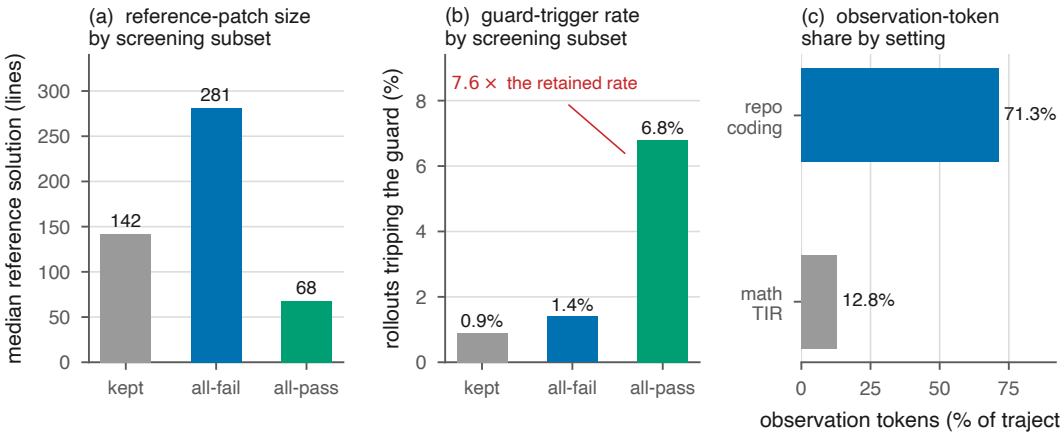


Figure 9: **Recovered-pool diagnostics. (a)** Median reference-patch sizes differ across screening subsets. **(b)** Guard-trigger rates are 0.9%, 1.4%, and 6.8% on retained, screening-all-fail, and screening-all-pass tasks, respectively. These do not measure all verifier defects. **(c)** Observation-token shares in the coding and TIR comparison.

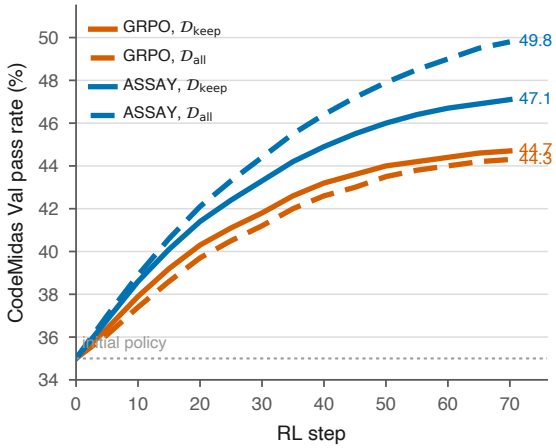


Figure 10: **CODEMIDAS Val during training.** All four runs start at 35.0. Final scores are 44.7 and 44.3 for GRPO, and 47.1 and 49.8 for ASSAY, on $\mathcal{D}_{\text{keep}}$ and $\mathcal{D}_{\text{all}}$, respectively. Lines connect the reported checkpoints; no statistical significance is inferred from the small GRPO difference.

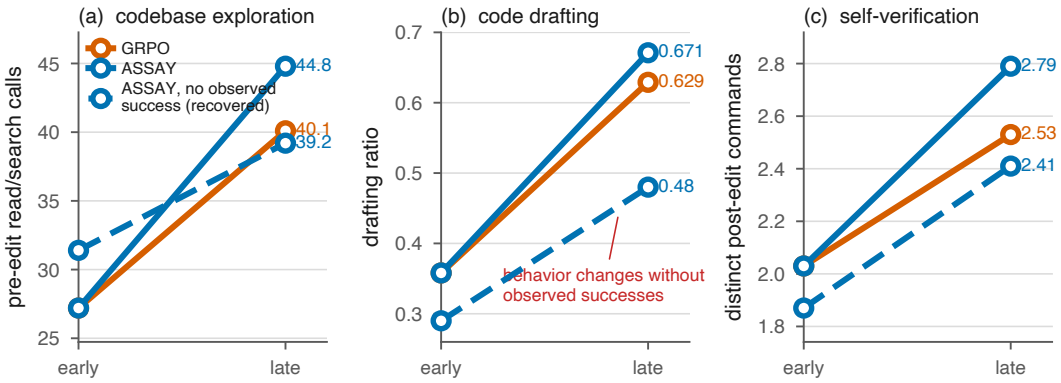


Figure 11: **Behavioral changes.** GRPO values are from Ye et al. (2026). The dashed series concerns recovered tasks with no observed successful rollout in the analyzed samples. Their behavior can change through both their own updates and cross-task transfer; this comparison does not isolate those effects.

J Reproducibility Details

Environment eligibility. Reproduction requires the 8,173 environments before final selection, task identifiers, screening outcomes, and quality-check artifacts. The published 5,545-task pool alone does not identify the recovered subset. Evaluation requires the same held-out task sets.

Value supervision. Table 6 gives the inventory. Reproduction also requires causal states, record-to-task mapping, outcome labels, and the distinction between repeated checks and agent trajectories. A pass/fail log without state-bearing records is insufficient. Critic-corpus membership for each RL-pool comparison should accompany run-level artifacts; point-score tables alone do not resolve all supervision-matching questions.

Optimization and evaluation. We adopt SAO's update components, with the configuration in Table 3, not all its original hyperparameters. Equation 4 specifies the detached coefficient. Faithful reproduction requires behavior log-probabilities, batch construction, evaluation-harness versions, and random seeds. The tables do not include seed-level intervals or a full hardware and initialization-cost breakdown, so our claims are limited accordingly.

Analytical and derived quantities. Equation 1, Table 5, and Figure 4 are closed-form calculations. Figure 5b contains differences of value estimates rather than measured policy advantages. These distinctions are necessary to reproduce the interpretation as well as the figures.