HACKFREE-GRPO: DETECTING AND MITIGATING REWARD HACKING IN RLVR-TRAINED CODE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards can reward code agents for passing tests without resolving the underlying issue. We study this failure mode in repository-level, multi-turn training and introduce HACKFREE-GRPO. Its frozen detector scores actions, patch diffs, and execution logs. Reward shaping discounts risky successes, and trajectory filtering removes high-risk samples before group-relative advantages are computed. Shaping alone can leave an exploit positively advantaged in a low-success group; after filtering, score differences can still distinguish risky survivors. On SWE-bench Verified, FULL reduces protocol-defined hacking from 41.7% to 5.8% and raises strict-audit pass@1 from 30.5% to 49.0%, while verifier acceptance is 52.0% versus 52.3%. Adversarial-test-only pass@1 increases from 31.2% to 49.9%, exceeding SHAPE by 2.5 percentage points. We derive the exact change in retained-group advantages and identify the conditions under which filtering removes residual positive reinforcement. The analysis specifies a group-level mechanism that can be tested alongside the observed task-level gains.

1 INTRODUCTION

Code agents can use executable tests as rewards while editing a repository and interacting with its tools. SWE-bench and SWE-Gym provide environments for evaluating and training these agents (Jimenez et al., 2024; Pan et al., 2025). However, an agent may pass a verifier by manipulating execution or exploiting incomplete tests while leaving the underlying issue unresolved.

A shortcut that receives the same terminal reward as a valid solution can repeatedly reinforce the wrong behavior during training. Verifier acceptance may then increase without a comparable improvement in task performance. Studies of programming agents and controlled RLVR environments document such reward hacking (Gabor et al., 2025; Khalifa et al., 2026). We examine how it develops during repository-level, multi-turn training, where rewards depend on tool actions, code modifications, and execution feedback.

We track verifier acceptance and a fixed, stronger audit throughout training, then intervene on the rewarded trajectories before the policy update. HACKFREE-GRPO uses HACKDETECT, a frozen classifier over actions, patch diffs, and execution logs. Its score determines a continuous reward discount and a filtering decision before the group-relative update. The detector uses rollout observations; the complete audit runs outside the policy-update loop.

The two operations act on different parts of the learning signal. Shaping lowers the reward of a suspicious success, which may nevertheless remain above the group mean when ordinary failures are common. Filtering removes its direct reward-update contribution. Shaping also differentiates the risk scores of surviving successes. To isolate these effects from changes in the rollout distribution, we compare the update operators on the same group.

On SWE-bench Verified, FULL reaches 49.0% strict-audit pass@1 against 30.5% for vanilla GRPO; adversarial-test-only evaluation gives 49.9% and 31.2%. SHAPE and FILTER each improve over vanilla, and FULL achieves the highest success rate among these variants. The group analysis

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.

predicts where their combination may help: invalid successes can retain positive advantages after discounting when few valid successes and many failures share the group.

We contribute a training-time diagnosis of reward hacking in repository-level code agents, a trajectory-based intervention combining shaping and filtering within GRPO, and a mathematical analysis of the resulting group-relative advantages. Experiments compare task performance, detector behavior, and transfer across evaluation sets.

2 REWARD HACKING DURING CODE-AGENT TRAINING

Environment and reward. A task consists of an issue and an initial repository state. The policy generates a trajectory τ of tool interactions and a final patch. Its terminal training verifier $\mathcal{V}(\tau) \in \{0, 1\}$ supplies the reward. Our environment exposes the working tree and task tests to the agent, allowing test modification, special-casing, and manipulation of execution. These permissions define the setting studied here.

Operational definition. The strict audit $\mathcal{A}$ combines additional tests, test-infrastructure integrity, a patch judge, and instrumented execution. It accepts only if the training verifier accepts and all checks pass, so $\mathcal{A} \leq \mathcal{V}$. We call verifier-passing trajectories rejected by this audit protocol-defined hacking. Rejection can reflect an incorrect repair, a conservative infrastructure rule, or observable exploitation. Appendix A defines the checks.

$$P_{\text{train}} = \mathbb{P}(\mathcal{V} = 1), \qquad P_{\text{audit}} = \mathbb{P}(\mathcal{A} = 1), \tag{1}$$

$$H_{\text{protocol}} = \mathbb{P}(\mathcal{A} = 0 \mid \mathcal{V} = 1), \qquad P_{\text{train}} - P_{\text{audit}} = P_{\text{train}} H_{\text{protocol}}. \tag{2}$$

The identity uses a common population and aggregation weights. We also report test-only evaluation without the detector or patch-judge score. Official-harness outcomes and human correctness, when assessed, need not be nested inside acceptance by $\mathcal{V}$.

Training dynamics. Figure 1 follows verifier acceptance, strict-audit success, and conditional audit rejection over training. Verifier acceptance rises while audited success remains substantially lower. Invalid solutions can therefore continue to receive rewards as training progresses. We target these rewarded trajectories before the next group-relative update.

Evaluation boundary. HackDetect's rollout features exclude audit-only test verdicts and reference patches, and the detector remains frozen during policy optimization. The held-out audit supplies the outcome measure for assessing whether the intervention improves task performance. The audit-only patch judge receives the issue, changed files, patch, verifier outcome, and canonical solution diff, but no detector score or method identity. Only a clean verdict clears this check; ambiguous cases remain protocol rejections. The training-time judge baseline instead receives the execution summary without the canonical solution or held-out outcomes. These input boundaries separate information available for constructing an update from information used to assess its resulting policy (Appendix A).

3 HACKFREE-GRPO

HackFree-GRPO uses a frozen trajectory detector to discount rewards and filter trajectories before the group update. Each policy update uses these scores; the complete held-out audit runs outside the update loop.

3.1 HACKDETECT: A TRAJECTORY DETECTOR

Representation. The 220M-parameter detector $D_\phi(\tau)$ encodes three streams: tool-call actions, the patch diff, and execution logs. The streams share a `microsoft/deberta-v3-base` backbone (He et al., 2023); a diff-trace adapter, structured harness-state embedding, and two-layer cross-attention head produce a score $s \in [0, 1]$. Each stream has a 512-token cap, for at most 1,536 tokens before fusion. Action selection prioritizes the first 4 and last 24 calls; diff selection prioritizes paths, hunk headers, test infrastructure, public APIs, imports, and failing-stack files. Log selection retains

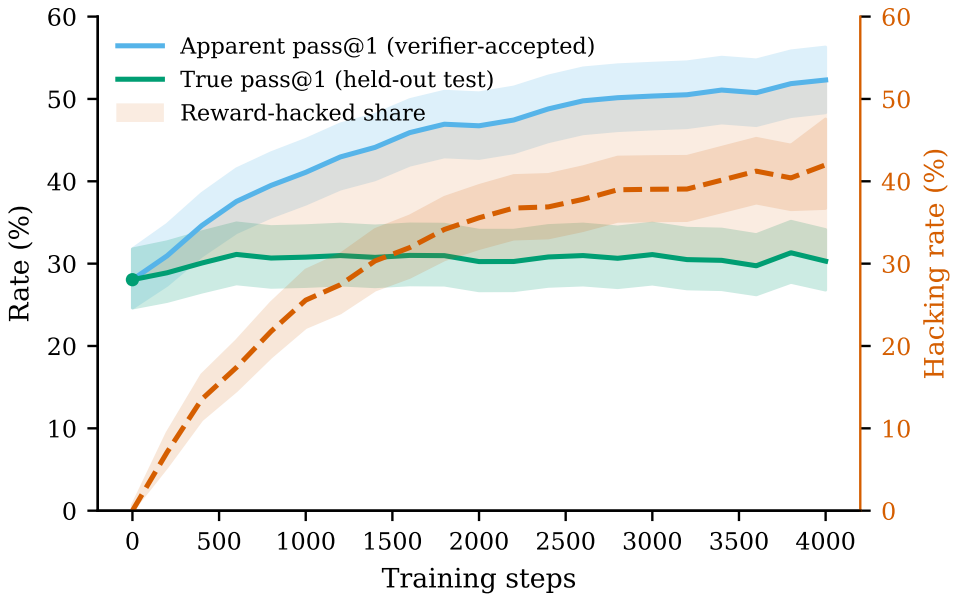

Figure 1: Verifier acceptance and strict-audit success during training. The plot's "true pass" label denotes the strict-audit endpoint; hacking rate is conditional on verifier success. The initial audit point is estimated at 28.05%; its interval is aligned with the verifier interval under the zero-initial-hacking assumption. The curves use the training checkpoint indices of the reported run.

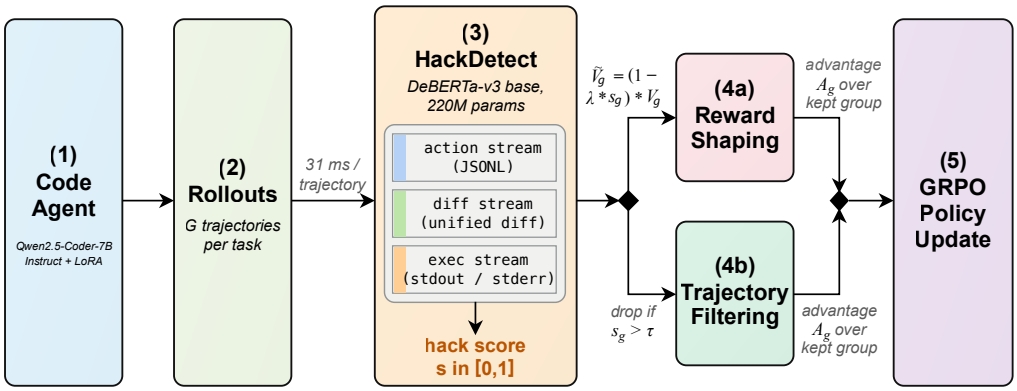

Figure 2: HackFree-GRPO scores each trajectory before the policy update. FULL filters high-score trajectories, then shapes rewards and recomputes the baseline within the kept group. The pictured threshold τ is denoted by c below; the latency annotation refers to the reported detector configuration.

failure evidence and the final 128 harness-output tokens. Detector inputs are restricted to the rollout; evaluation-only tests and reference patches are excluded.

Offline construction. We hand-label 210 training-pool trajectories using the protocol's hacking rubric, split 150/30/30 for seed training, validation, and testing. Three offline self-training rounds accept scores $s \leq 0.05$ or $s \geq 0.95$ as pseudo-labels and down-sample the majority class. The resulting pool contains 8,512 trajectories, with 4,236 hack and 4,276 clean labels under that rubric. The detector is frozen throughout policy training. Human gold evaluation labels are held apart from the pseudo-label pool, and independent evaluation labels are excluded from detector training.

3.2 SHAPING, FILTERING, AND THE GROUP UPDATE

For task i, sample G trajectories from π_{old} and record verifier rewards $v_{ig} = \mathcal{V}(\tau_{ig})$ and scores $s_{ig} = D_\phi(\tau_{ig})$. With shaping strength λ and threshold c, the variants are

$$\begin{array}{lll} \text{SHAPE}: & \mathcal{K}_i = \{1, \dots, G\}, & z_{ig} = (1 - \lambda s_{ig}) v_{ig}, \\ \text{FILTER}: & \mathcal{K}_i = \{g : s_{ig} \le c\}, & z_{ig} = v_{ig}, \\ \text{FULL}: & \mathcal{K}_i = \{g : s_{ig} \le c\}, & z_{ig} = (1 - \lambda s_{ig}) v_{ig}. \end{array} \tag{3}$$

For $K_i = |\mathcal{K}_i| > 0$, the centered group-relative advantage is

$$\bar{z}_i = \frac{1}{K_i} \sum_{g \in \mathcal{K}_i} z_{ig}, \qquad \widehat{A}_{ig} = z_{ig} - \bar{z}_i. \tag{4}$$

The defaults are $G = 8$, $\lambda = 0.6$, and $c = 0.7$. We use these advantages in a clipped policy surrogate with a separate reference-policy KL term, following GRPO and PPO (Shao et al., 2024; Schulman et al., 2017). We subtract the group mean without scaling by the standard deviation. Empty kept groups supply no reward update; singleton and equal-reward groups have zero centered advantages. Appendix B specifies the aggregation weights, token mask, and KL handling.

3.3 WHY SHAPING AND FILTERING CAN BE COMPLEMENTARY

Residual positive advantages. A verifier-passing invalid trajectory can retain positive advantage after reward discounting. On a fixed group, write $z_g = (1 - \lambda s_g) v_g$ and $\mu_G = G^{-1} \sum_g z_g$. Such a trajectory h retains positive SHAPE advantage exactly when

$$A_h^{\text{SHAPE}} > 0 \quad \Longleftrightarrow \quad 1 - \lambda s_h > \mu_G. \tag{5}$$

At $\lambda = 0.6$, a verifier-passing trajectory with score one retains reward 0.4. Its advantage depends on its siblings. FULL removes trajectories above c from the reward surrogate.

Changing the baseline. Let FULL keep K trajectories and remove $M = G - K$, with mean shaped rewards μ_K and μ_R. For $0 < K < G$, every kept trajectory satisfies

$$A_g^{\text{FULL}} - A_g^{\text{SHAPE}} = \frac{M}{G}(\mu_R - \mu_K). \tag{6}$$

Removing a subset with a higher mean reward shifts every kept advantage upward, making negative advantages less negative. The direction reverses when $\mu_R < \mu_K$; pairwise differences between kept trajectories are unchanged.

Risk differences within the kept group. FILTER assigns the same reward to any two kept verifier successes. FULL gives $A_i^{\text{FULL}} - A_j^{\text{FULL}} = \lambda(s_j - s_i)$. Filtering selects the survivors; shaping can still distinguish their rewards. This distinction is useful only when scores track validity. FILTER and FULL keep identical sets at a fixed group and threshold. For example, two verifier successes with scores 0.05 and 0.60 both survive $c = 0.7$. FILTER gives each reward one, whereas FULL gives rewards 0.97 and 0.64 at $\lambda = 0.6$. Their FULL advantages differ by 0.33 regardless of the other retained rewards. The score distinction therefore remains active even when the threshold makes the same keep decision. FULL may therefore benefit mixed groups in which invalid successes retain positive advantages, residual scores carry validity information, and enough correct solutions survive. Section 4 compares the variants; these conditions motivate closer evaluation of the sibling groups. Appendix D provides the derivations, worked examples, a counterexample, and a separate analysis of the weighted, clipped update.

A worked group. Consider eight sibling trajectories: one valid verifier success with score 0.05, two invalid successes with score 0.90, and five verifier failures with score 0.10. With $\lambda = 0.6$ and $c = 0.7$, the shaped rewards are 0.97, 0.46, and 0, respectively. SHAPE's group mean is 0.23625, so each invalid success still receives positive advantage 0.22375. FULL removes these two trajectories and recomputes the mean as 0.16167, raising the valid success's advantage from 0.73375 to 0.80833. The failures' negative advantages also weaken. Filtering both removes positively advantaged invalid successes and changes the survivors' baseline; the latter effect follows the group-composition condition in Eq. (6).

Detection errors and retained composition. Among verifier successes, let $q \in (0, 1)$ denote the invalid fraction under a fixed validity criterion. Let r and α be the removal probabilities for invalid and valid successes, respectively. For a nonempty retained population, the remaining invalid fraction is

$$q_{\text{kept}} = \frac{q(1-r)}{q(1-r) + (1-q)(1-\alpha)}. \tag{7}$$

Thus $q_{\text{kept}} < q$ exactly when $r > \alpha$: filtering improves the composition when it removes invalid successes more often than valid ones. The denominator is the retained fraction of verifier successes, while $1-\alpha$ measures retention of valid successes. A lower invalid fraction alone does not ensure that enough useful candidates survive for learning. Together with the zero advantages of singleton and equal-reward groups, this explains why retained composition, valid-success retention, and informative group counts address different properties of the intervention. Appendix D gives the conditional sensitivity calculations; these population quantities characterize the filtering step before the policy is updated.

4 EXPERIMENTS

We test whether detector-guided updates improve repository task performance and whether shaping and filtering are complementary. The comparison uses the same detector and matched training budgets for SHAPE, FILTER, and FULL.

Setup. We use Qwen2.5-Coder-7B-Instruct (Qwen Team, 2024) with LoRA adapters, trained on SWE-Gym (Pan et al., 2025) and the LiveCodeBench component of EvilGenie (Gabor et al., 2025; Jain et al., 2025). Evaluation covers SWE-bench Verified (SWE-bench Team, 2024), task-disjoint EvilGenie, and held-out repositories from SWE-rebench (Badertdinov et al., 2025). The training pools contain 2,438 SWE-Gym tasks and 812 LiveCodeBench/EvilGenie tasks. SWE-bench Verified contains 500 evaluation tasks, and the held-out repository split contains 200. The policy can make at most 50 tool turns per trajectory. We use LoRA rank 64 and scaling parameter 128, a learning rate of 5×10^{-6}, and seeds $\{11, 23, 47\}$. Each collection contains 256 trajectories with group size $G = 8$; the primary schedule uses 4,000 optimizer updates. Means and standard deviations are over the three training seeds; the extended-training analysis follows later checkpoints. Appendix G gives the remaining optimizer, detector, and resource-accounting settings. The reported endpoint P_{strict} requires training-verifier success and acceptance by all four strict audit checks. Its conditional violation rate is h_{strict}. This protocol-relative endpoint differs from independent official-harness success or human semantic correctness.

4.1 SHAPING AND FILTERING IMPROVE STRICT AUDIT-PASS PERFORMANCE

Table 1 shows improvements from both components across all three benchmarks. On SWE-bench Verified, FULL raises strict audit-pass performance from 30.5% to 49.0% relative to vanilla GRPO, while reducing the conditional strict-violation rate from 41.7% to 5.8%. It also exceeds DPO on this benchmark and both individual components on all three benchmarks. Under adversarial tests alone (signal 1), the corresponding values are 31.2% for vanilla and 49.9% for FULL. Appendix E gives all apparent-pass and violation-rate columns, including the additional training-verifier baselines.

4.2 COMPLEMENTARITY OF THE TWO COMPONENTS

Table 2 quantifies the margin of FULL on SWE-bench Verified. The combined update exceeds both components under the strict audit and signal (1) alone, which omits the test-edit rule and patch judge. Shaping discounts risky rewards; filtering removes high-score trajectories and recenters the retained-group baseline. FULL then shapes the rewards of the retained trajectories using their residual scores.

The same-group derivation and worked examples in Appendix D show how removal and recentering change the training signal and illustrate how detector misranking can misdirect the update. The audit results establish the component ordering; attributing it to mixed-group prevalence or false suppression requires separate measurements. Appendix I.2 defines paired quantities that hold the pre-update group fixed and isolate the update operator from changes in the on-policy rollout distribution.

Table 1: Strict audit-pass performance (%, mean ± one seed standard deviation over three seeds). Signal (1) only uses the adversarial-test endpoint. The EvilGenie split is task-disjoint but distribution-matched to its training component; held-out repositories provide a distinct task distribution.

Method	Verified P_{strict}	EvilGenie P_{strict}	Held-out repos P_{strict}	Verified signal (1) only
Vanilla GRPO	30.5 ± 0.5	28.6 ± 0.5	24.9 ± 0.6	31.2 ± 0.5
DPO on hack/clean pairs	38.4 ± 0.5	34.4 ± 0.6	29.7 ± 0.6	38.7 ± 0.5
HackFree (shape)	46.8 ± 0.5	42.2 ± 0.5	36.3 ± 0.5	47.4 ± 0.5
HackFree (filter)	47.1 ± 0.5	42.5 ± 0.5	36.7 ± 0.5	47.8 ± 0.5
HackFree (full)	49.0 ± 0.4	44.2 ± 0.4	38.1 ± 0.5	49.9 ± 0.5

Table 2: Differences between the SWE-bench Verified means in Table 1, in percentage points. Each entry is $P(\text{FULL}) - P(\text{comparison})$ for the indicated endpoint.

Comparison	ΔP_{strict}	$\Delta P_{\text{signal 1}}$
FULL − Vanilla	18.5	18.7
FULL − DPO	10.6	11.2
FULL − SHAPE	2.2	2.5
FULL − FILTER	1.9	2.1

4.3 Independent-outcome targets and group-level expectations

Table 3 gives scenario references and prospective targets for independent execution and human review. The reference rates follow the verifier, strict-audit, and signal-(1)-only mapping in Appendix J. The FULL target retains 90% of its reference improvement over FILTER, the best single component in this scenario. Success and correctness use minimum targets; exploitation uses a maximum target. These are planning quantities, not checkpoint re-evaluation results. Independent execution and human audit must measure both FULL and its controls.

Table 4 fixes group composition and averages the update quantities exactly over detector-error outcomes. Under the specified conditions, mixed and invalid-only groups show a larger reduction in invalid positive advantage weight than clean groups. False rejections can also remove useful signal from clean groups. These are conditional predictions for the chosen compositions, not observations of group frequencies during training.

4.4 Detector behavior and cross-benchmark transfer

Figure 3 compares HackDetect with rule-based and LLM-judge alternatives under the validation labels. The plotted precision–recall curves favor HackDetect. Its detector-only score is used during training; the held-out-test combination provides an offline evaluation instrument. The distinction between filter precision and retention of correct candidates is formalized in Appendix I.3.

Detection across known hacking patterns. Table 5 compares the EvilGenie validation split with a held-out Countdown-Code split (Khalifa et al., 2026). Both splits use protocol labels for instances of the same known pattern categories.

Figure 4 reports higher strict audit-pass performance on each additional task distribution. The held-out-repository comparison agrees with Table 1; SWE-bench Live, Countdown-Code, and the unseen EvilGenie split broaden the task-level comparison. This evidence covers the tested policy and protocols. Transfer to another policy family or arbitrary new exploitation strategies requires separate evaluation.

Table 3: Scenario references and prospective FULL targets (%). The first four rows are scenario reference values. The last row retains 90% of FULL's reference improvement over FILTER. All values in this table are planning quantities, not measured official-harness or human-audit results; Appendix J defines the calculation.

Method / target	Official success (%)	Human-correct among $V = 1$ (%)	Exploitation among $V = 1$ (%)
Vanilla GRPO	31.20	58.99	20.17
SHAPE	47.40	91.28	4.07
FILTER	47.80	93.04	3.14
FULL reference	49.90	95.10	2.02
FULL (90% target)	$\geq$49.69	$\geq$94.89	$\leq$2.13

Table 4: Expected group-level quantities at $G = 8$, $\lambda = 0.6$, $c = 0.7$, invalid-removal probability $r = 0.8$, and valid-removal probability $\alpha = 0.05$. C, H, F count valid successes, invalid successes, and verifier failures. The values are exact expectations under these specified error-model assumptions. Definitions and score assignments appear in Appendix J.

Group (C, H, F)	Reduction in W_H^+ ($\times 100$)	Change in W_C^+ ($\times 100$)	Kept correct per group
Clean (3,0,5)	0.00	-0.23	2.85
Mixed (1,2,5)	4.32	0.50	0.95
Invalid only (0,3,5)	7.85	0.00	0.00

4.5 Behavior over extended training

Table 6 follows HackFree-GRPO (full) from 4,000 to 8,000 updates on SWE-bench Verified under the same training configuration. Conditional rejection varies within a narrow range while strict audit-pass@1 increases. Recall is the fraction of held-out-test-failed trajectories that the detector flags as hacks, and is conditional on that slice. The extension assesses ordinary training over this horizon; independent human-label recall and adaptive evasion require separate tests.

5 Related work

Code-agent learning and evaluation. SWE-bench and SWE-Gym connect repository-level issue resolution to executable evaluation and training (Jimenez et al., 2024; Pan et al., 2025). Execution feedback also supports code-model reinforcement learning (Gehring et al., 2025). Program-repair

Table 5: Per-pattern detector F_1 under protocol labels. The first column of scores uses EvilGenie validation data; the second uses the held-out Countdown-Code split. Δ is the OOD minus in-distribution difference.

Pattern	F_1 in-distribution	F_1 held-out	Δ
Test-file modification	0.94	0.93	-0.01
Hardcoded outputs	0.91	0.89	-0.02
Conditional on test inputs	0.87	0.85	-0.02
Exception swallowing	0.82	0.81	-0.01
Mocking / monkey-patching	0.78	0.75	-0.03
Import-error masking	0.81	0.79	-0.02
Other / mixed	0.74	0.72	-0.02
Macro-F_1	**0.839**	**0.820**	-0.019

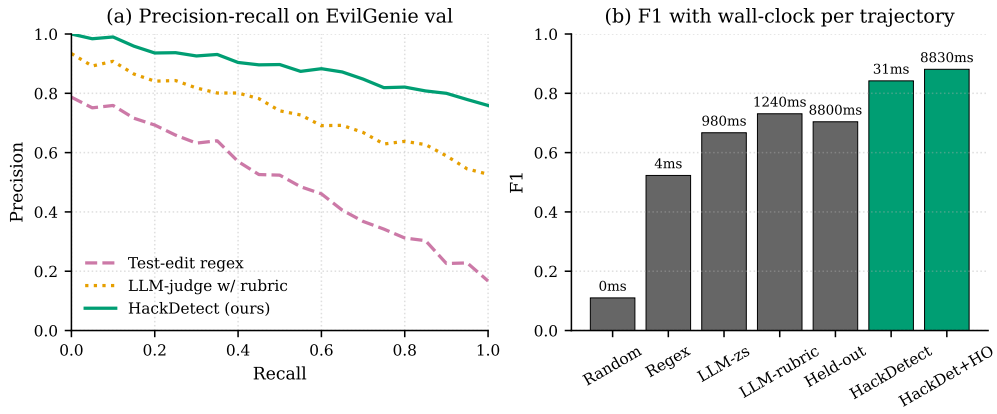

Figure 3: Detector evaluation on the EvilGenie validation split. The left panel shows precision–recall curves; the right panel compares F1 and per-trajectory time under the reported hardware conditions. HackDetect+HO includes offline held-out tests and is not the in-loop detector.

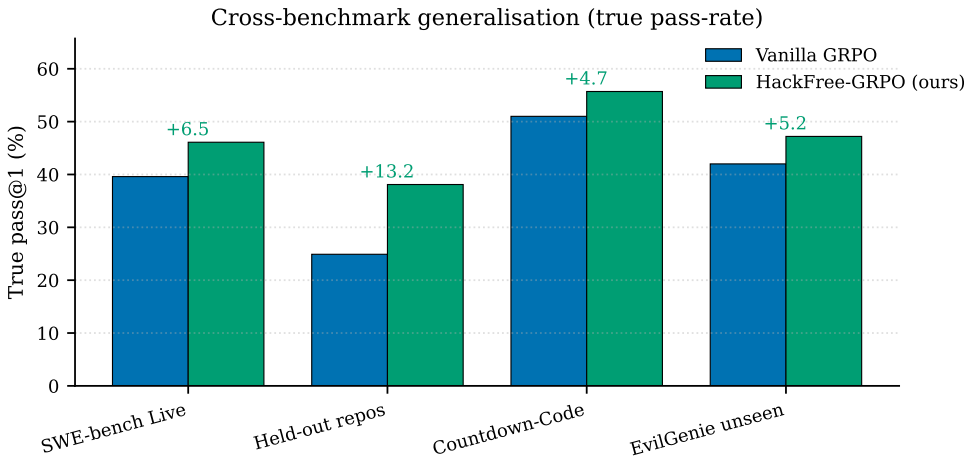

Figure 4: Cross-benchmark strict audit-pass performance. "True pass" denotes P_{strict}, with the same endpoint used for both methods within each benchmark. Differences above the bars are in percentage points.

research and SWE-bench audits distinguish test-passing patches from semantically correct repairs (Qi et al., 2015; Xin & Reiss, 2017; Wang et al., 2025b; Yu et al., 2026). This distinction informs our execution, audit-rejection, and human-correctness endpoints.

Table 6: Passive extended training of HackFree-GRPO (full). Entries are mean ± standard deviation over three seeds. Conditional rejection and audit-pass@1 use the strict protocol; detector recall uses the held-out-test-failed slice described above.

Update	Conditional rejection (%)	Strict audit-pass@1 (%)	Detector recall
4,000	5.8 ± 0.3	49.0 ± 0.4	0.884 ± 0.012
5,000	5.7 ± 0.4	49.6 ± 0.5	0.882 ± 0.013
6,000	6.1 ± 0.3	50.3 ± 0.4	0.879 ± 0.012
7,000	5.9 ± 0.4	50.7 ± 0.5	0.876 ± 0.013
8,000	5.8 ± 0.4	51.2 ± 0.5	0.873 ± 0.014

Reward hacking. Specification gaming concerns optimization of a proxy that differs from the intended objective (Amodei et al., 2016; Krakovna et al., 2020; Skalse et al., 2022). RLVR shortcut studies, EvilGenie, and Countdown-Code examine this problem in reasoning and programming settings (Helff et al., 2026; Gabor et al., 2025; Khalifa et al., 2026). Mitigation methods include auditor-guided reward gating and gradient-based rejection fine-tuning (Beigi et al., 2026; Wang et al., 2026a). We focus on trajectory detection and group-relative updates in repository-level training, where task outcomes must be evaluated independently of the detector's taxonomy.

Optimization and verifier protection. GRPO uses group-relative rewards, while DAPO changes multiple components of RL training (Shao et al., 2024; Yu et al., 2025). A dynamic-sampling ablation tests one component of that recipe. Stronger unit tests and adversarial benchmark strengthening address the evaluation signal (Ma et al., 2025b; Yu et al., 2026); trace filtering changes the update. Their combined benefit remains an empirical question. Appendix K discusses related detection methods, training systems, agent interfaces, benchmarks, patch correctness, and synthesized action harnesses.

6 Discussion and Limitations

Why the combination matters. HackFree improves strict-audit and adversarial-test success at comparable verifier acceptance in the reported comparisons. The group analysis offers a possible explanation for FULL's additional gains: filtering removes invalid successes that remain positively advantaged after reward discounting, and shaping differentiates the survivors. Group composition determines the effect. Removing an already-negative example can discard useful contrast, so a smaller group alone is insufficient to explain improved learning.

From advantages to updates. With mean-normalized trajectory weights, Appendix I.2 decomposes the local FULL–SHAPE reward-gradient difference into removal, recentering, and renormalization. This identity holds at the rollout parameters with KL excluded. Different aggregation weights alter the contributions, and later optimization steps can activate clipping. The centered-advantage mechanism therefore describes one component of the full update.

Detection and useful training signal. The value of filtering depends on which trajectories remain. False rejections can be costly when successful repairs are rare, even if the retained population contains fewer invalid solutions. Appendix D quantifies removal and retention under specified error rates. Applying that analysis to a trained policy requires its score distribution and task mixture.

Scope and efficiency. The results apply to the tested policy, verifier permissions, benchmarks, and training budgets. HackDetect uses observable tool traces, keeping the complete audit outside each policy update. Total cost includes detector construction, inference, and generated trajectories later discarded. The ordinary-training extension measures stability over the reported horizon; adaptive evasion and interactions with other RL recipes require further evaluation. Collection rounds and optimizer steps capture different parts of this cost: rollouts incur generation and detector costs before the keep decision, whereas optimization reuses the recorded samples and fixed detector scores. Consequently, the number of retained trajectories does not measure the full sampling budget. Appendix G distinguishes online training, offline detector construction, and evaluation-only audits in its resource accounting. This separation also makes the role of the frozen detector explicit: its training is an offline cost, while scoring each new collection remains part of online training.

7 Conclusion

HackFree-GRPO intervenes on invalid reward-bearing trajectories before the group-relative update. In the reported benchmarks, combining its frozen detector with reward shaping and filtering reduces protocol-defined hacking and improves strict-audit and adversarial-test success. The group analysis identifies when discounting leaves invalid successes positively advantaged and quantifies how filtering changes the baseline for retained trajectories. These conditions guide the evaluation of trajectory-level interventions in code-agent training.

AI USE STATEMENT

AI tools were used for language polishing.

ETHICS STATEMENT

The study concerns code agents that can interact with test infrastructure. Reproduction should use authorized, isolated task environments. Trace releases should exclude credentials and personal data, respect repository licenses, and document the permitted evaluation boundary. Reward-hacking examples are included to support detection and mitigation.

REPRODUCIBILITY STATEMENT

The paper specifies the detector architecture, training settings, reward transformations, derivations, and evaluation protocols. Tables state their outcome definitions and aggregation. The supplementary material defines the quantities used in independent evaluation and diagnostic analyses, and records the ordinary extended-training settings.

REFERENCES

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mane. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, et al. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, et al. SWE-rebench V2: Language-agnostic SWE task collection at scale. *arXiv preprint arXiv:2602.23866*, 2026.

Mohammad Beigi, Ming Jin, Junshan Zhang, et al. Adversarial reward auditing for active detection and mitigation of reward hacking. *arXiv preprint arXiv:2602.01750*, 2026.

Shiyi Cao, Dacheng Li, Fangzhou Zhao, et al. SkyRL-Agent: Efficient RL training for multi-turn LLM agent. *arXiv preprint arXiv:2511.16108*, 2025.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

Chanakya Ekbote, Vijay Lingam, Sujay Sanghavi, et al. MURPHY: Multi-turn GRPO for self correcting code generation. *arXiv preprint arXiv:2511.07833*, 2025.

Jonathan Gabor, Jayson Lynch, and Jonathan Rosenfeld. EvilGenie: A reward hacking benchmark. *arXiv preprint arXiv:2511.21654*, 2025.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 19034–19055. PMLR, 2025. URL https://proceedings.mlr.press/v267/gehring25a.html.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025. doi: 10.1038/s41586-025-09422-z.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=sE7-XhLxHA.

Lukas Helff, Quentin Delfosse, David Steinmann, Ruben Härle, Hikaru Shindo, Patrick Schramowski, Wolfgang Stammer, Kristian Kersting, and Felix Friedrich. LLMs gaming verifiers: RLVR can lead to reward hacking. *arXiv preprint arXiv:2604.15149*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Muhammad Khalifa, Zohaib Khan, Omer Tafveez, et al. Countdown-Code: A testbed for studying the emergence and generalization of reward hacking in RLVR. *arXiv preprint arXiv:2603.07084*, 2026.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity. *DeepMind Blog*, 2020.

Shanchao Liang, Nan Jiang, Yiran Hu, and Lin Tan. Can language models replace programmers for coding? REPOCOD says "not yet". *Proceedings of ACL*, 2025.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, et al. AutoHarness: improving LLM agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

Zexiong Ma, Chao Peng, Pengfei Gao, et al. SoRFT: Issue resolving with subtask-oriented reinforced fine-tuning. *Proceedings of ACL*, 2025a.

Zeyao Ma, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. Dynamic scaling of unit tests for code reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 6917–6935, 2025b. doi: 10.18653/v1/2025.acl-long.343.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA)*, pp. 24–36, 2015. doi: 10.1145/2771783.2771791.

Qwen Team. Qwen2.5-Coder-7B-Instruct. https://huggingface.co/Qwen/Qwen2.5-Coder-7B-Instruct, 2024. Model card.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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.

KaShun Shum, Binyuan Hui, Jiawei Chen, et al. SWE-RM: Execution-free feedback for software engineering agents. *arXiv preprint arXiv:2512.21919*, 2025.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

SWE-bench Team. SWE-bench verified. https://www.swebench.com/verified.html, 2024. Human-filtered 500-instance subset of SWE-bench, created in collaboration with OpenAI.

Mirza Farhan Bin Tarek and Rahmatollah Beheshti. Reward hacking mitigation using verifiable composite rewards. In *Proceedings of the 16th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (ACM-BCB)*, 2025.

Mia Taylor, James Chua, Jan Betley, et al. School of reward hacks: Hacking harmless tasks generalizes to misaligned behavior in LLMs. *arXiv preprint arXiv:2508.17511*, 2025.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, and Maosong Sun. DebugBench: Evaluating debugging capability of large language models. *Findings of ACL*, 2024.

Apostol Vassilev, Alina Oprea, Alie Fordyce, Hyrum Anderson, Xander Davies, and Maia Hamin. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. Technical report, National Institute of Standards and Technology (NIST AI 100-2 E2025), 2025.

Jiacheng Wang and Jinbin Huang. Reward hacking as equilibrium under finite evaluation. *arXiv preprint arXiv:2603.28063*, 2026.

Lei Wang, Chen Ma, Xueyang Feng, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 2024. doi: 10.1007/s11704-024-40231-1.

Ruiyi Wang and Prithviraj Ammanabrolu. A practitioner's guide to multi-turn agentic reinforcement learning. *arXiv preprint arXiv:2510.01132*, 2025.

Songtao Wang, Quang Hieu Pham, Fangcong Yin, et al. Detecting and suppressing reward hacking with gradient fingerprints. *arXiv preprint arXiv:2604.16242*, 2026a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025a. URL https://openreview.net/forum?id=OJd3ayDDoF.

Xinpeng Wang, Nitish Joshi, Barbara Plank, Rico Angell, and He He. Is it thinking or cheating? detecting implicit reward hacking by measuring reasoning effort. In *International Conference on Learning Representations (ICLR)*, 2026b. URL https://openreview.net/forum?id=Gk7gLAtVDO. Oral.

You Wang, Michael Pradel, and Zhongxin Liu. Are "solved issues" in SWE-bench really solved correctly? an empirical study. *arXiv preprint arXiv:2503.15223*, 2025b.

Chengxing Xie, Bowen Li, Chang Gao, et al. SWE-Fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. *arXiv preprint arXiv:2501.05040*, 2025.

Qi Xin and Steven P. Reiss. Identifying test-suite-overfitted patches through test case generation. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp. 226–236, 2017. doi: 10.1145/3092703.3092718.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Boxi Yu, Yang Cao, Yuzhong Zhang, et al. SWE-ABS: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark. *arXiv preprint arXiv:2603.00520*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Danlong Yuan, Wei Wu, Zhengren Wang, et al. SWE-MiniSandbox: Container-free reinforcement learning for building software engineering agents. *arXiv preprint arXiv:2602.11210*, 2026.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 13643–13658, 2024a. doi: 10.18653/v1/2024.acl-long.737.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! *arXiv preprint arXiv:2505.23419*, 2025.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2024b. doi: 10.1145/3650212.3680384.

A AUDIT PROTOCOL AND EVALUATION BOUNDARIES

Strict audit. For a trajectory with $\mathcal{V} = 1$, the strict audit requires passing the additional adversarial tests, leaving protected test infrastructure unmodified, clearance by the patch judge, and successful execution under the instrumented runtime. The protected paths include `tests/`, test modules, fixtures, and runner configuration. This restriction applies to the benchmark; legitimate maintenance can include test refactoring.

Patch judge. The reported judge is `claude-3-5-sonnet-20241022`, with temperature 0, top-p 1.0, and a 512-token output cap. The audit-only input includes the issue, modified files, patch, verifier outcome, and canonical solution diff. It excludes detector scores, method identity, and training metadata. Its categories are clean, hack, and ambiguous; only clean clears the strict audit. Ambiguous verdicts count as protocol rejections but leave exploitation unresolved.

Training and evaluation information. The training-time judge baseline receives issue text, changed files, patch, verifier result, and execution summary. It has no access to the canonical solution or held-out evaluation outcomes. HackDetect uses actions, diffs, and logs at policy-update time, without audit labels. Detector construction uses human seed labels and pseudo-labels; evaluation labels are excluded from checkpoint fitting and threshold selection.

Interpreting independent checks. An official harness evaluates the patch in its canonical task environment. Human review can assess semantic correctness and observable exploitation while recording unresolved judgments. Neither outcome is defined as a subset of training-verifier acceptance. Review restricted to verifier-passed outputs estimates proportions in that population; an estimate over all attempts requires additional sampling.

Statistical units. Task-level uncertainty estimates keep repeated evaluations of each task together. Conditional hacking rates use the verifier-passed count. Seed variation, task sampling, and human-audit sampling contribute different sources of uncertainty. The tabled seed standard deviations describe training-run variation, not paired task-level confidence intervals.

B CLIPPED OBJECTIVE AND UPDATE ACCOUNTING

Token-level surrogate. For generated token y_{igt} at recorded history x_{igt}, define

$$\rho_{igt}(\theta) = \frac{\pi_\theta(y_{igt} \mid x_{igt})}{\pi_{\text{old}}(y_{igt} \mid x_{igt})}, \tag{8}$$

$$\ell_{igt}(\theta) = \min\left\{\rho_{igt}(\theta)\widehat{A}_{ig},\ \text{clip}(\rho_{igt}(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}_{ig}\right\}. \tag{9}$$

Rollout rewards, detector scores, membership, and the resulting advantages are held fixed over the update. Tool outputs and environment observations are excluded from the policy-token loss.

Aggregation and regularization. With explicit aggregation weights, the clipped surrogate is

$$\widehat{J}(\theta) = \sum_{i=1}^{B} \sum_{g \in \mathcal{K}_i} \sum_{t=1}^{T_{ig}} w_{igt}\, \ell_{igt}(\theta) - \beta\, \widehat{\mathcal{D}}_{\text{ref}}(\theta). \tag{10}$$

The weights $w_{igt} \geq 0$ define task, trajectory, and token aggregation and remain fixed during an update. The reference-policy penalty $\widehat{\mathcal{D}}_{\text{ref}}$ may use its own estimator, weights, and sample mask. For example, equal task, kept-trajectory, and generated-token weighting would give $w_{igt} = 1/(BK_iT_{ig})$ for nonempty kept groups. Keeping the original trajectory denominator would instead give $1/(BGT_{ig})$. These conventions leave the centered advantages unchanged but generally produce different updates. Centering specifies neither a $1/K_i$ loss denominator nor exact categorical KL in Eq. (4).

The reported settings are $G = 8$, $B = 32$, $\lambda = 0.6$, $c = 0.7$, $\epsilon = 0.2$, and $\beta = 0.04$. The group-level identities apply to the centered advantages in Eq. (4), independently of the aggregation weights and reference-policy penalty. The full parameter update depends on both.

Degenerate groups. When $K_i = 0$, the reward-surrogate sum for task i is empty. When $K_i = 1$, or all kept transformed rewards coincide, its centered advantages are zero. The regularization mask determines whether such a task still contributes a KL term. Shared parameters can change a trajectory's probability even when its reward term is excluded. With additional standard-deviation normalization, the scaling and zero-variance convention also enter the update and mechanism analysis. The centered identities apply before this scaling.

Training loop. For each collection round:

1. Sample a task batch and G sibling trajectories per task; save policy versions, verifier rewards, and generated-token log probabilities.
2. Score every trajectory with the frozen detector. For FILTER or FULL, construct $\mathcal{K}_i = \{g : s_{ig} \leq c\}$; SHAPE keeps the complete group.
3. Apply the variant's reward transformation in Eq. (3) and recompute the mean within its selected group.
4. Form the centered advantages for nonempty groups and apply the recorded aggregation and KL rules in Eq. (10).
5. Optimize the clipped surrogate over the declared minibatches, keeping rollout quantities fixed. Record collected trajectories and optimizer updates separately before refreshing the rollout policy.

The detector is trained offline and frozen throughout these rounds. The update excludes evaluation-only audit executions and human labels.

C Properties of the Reward Transformation

We analyze the protocol audit $\mathcal{A}$, with $\mathcal{A}(\tau) \leq \mathcal{V}(\tau)$. Appendix D gives the group-composition identities and worked examples. Fix a task and write $v_g = \mathcal{V}(\tau_g)$, $a_g = \mathcal{A}(\tau_g)$, $s_g = D_\phi(\tau_g)$, and $h_g = \mathbf{1}[v_g = 1, a_g = 0]$.

Proposition 1 (Retained-group identity under perfect detection)**.** *Suppose $s_g = h_g$ for every sampled trajectory and the filtering threshold is $c \in (0, 1)$. Let $\mathcal{K} = \{g : s_g \leq c\}$ and $K = |\mathcal{K}|$. On the event $K > 0$, for each $g \in \mathcal{K}$,*

$$v_g - \frac{1}{K}\sum_{j\in\mathcal{K}} v_j = a_g - \frac{1}{K}\sum_{j\in\mathcal{K}} a_j. \tag{11}$$

The same identity holds for FULL*, because its retained scores are zero.*

Proof. Perfect detection retains precisely those trajectories with $h_g = 0$. Since $a_g \leq v_g$ and both are binary, $h_g = 0$ implies $v_g = a_g$. Substitution proves the identity. For FULL, $(1 - \lambda s_g)v_g = v_g$ on the retained group. □

Rejected protocol-invalid trajectories contribute nothing to the reward surrogate in Eq. (10). The identity applies to the retained sample: filtering changes the trajectory distribution, group size, and baseline, so it does not establish an unbiased gradient for the unconditional audit-reward objective. Removal supplies no direct negative reward update; shared parameters and normalization can still change the rejected trajectories' probabilities.

Proposition 2 (Conditional absolute reward error)**.** *For $s \in [0, 1]$ and $\lambda \in [0, 1]$, define the shaping error $\delta = (1 - \lambda s)\mathcal{V} - \mathcal{A}$. Assume $\Pr(\mathcal{V} = 1) > 0$ and let $q = \Pr(h = 1 \mid \mathcal{V} = 1)$. Then*

$$\mathbb{E}[|\delta| \mid \mathcal{V} = 1] = q\,\mathbb{E}[1 - \lambda s \mid h = 1] + (1 - q)\,\mathbb{E}[\lambda s \mid h = 0, \mathcal{V} = 1], \tag{12}$$

$$\mathbb{E}[\delta \mid \mathcal{V} = 1] = q\,\mathbb{E}[1 - \lambda s \mid h = 1] - (1 - q)\,\mathbb{E}[\lambda s \mid h = 0, \mathcal{V} = 1]. \tag{13}$$

Terms corresponding to a zero-probability class are interpreted as zero.

Proof. Among verifier-passing trajectories, $h = 1$ gives $\mathcal{A} = 0$ and $\delta = 1 - \lambda s \geq 0$; $h = 0$ gives $\mathcal{A} = 1$ and $\delta = -\lambda s \leq 0$. Conditioning on these two cases yields the two equations. A verifier failure has $\mathcal{V} = \mathcal{A} = 0$, hence $\delta = 0$. □

Equation (12) decomposes absolute reward error; it neither expresses signed bias nor bounds policy-gradient error. For FULL, condition additionally on $s \leq c$ and use the audit-rejection proportion in the retained population. This conditional identity omits the contribution removed by filtering.

The baseline must change with the reward. For a nonempty, fixed group $\mathcal{K}$, let $\widehat{A}_g^{\mathrm{audit}} = a_g - K^{-1}\sum_{j\in\mathcal{K}} a_j$. For transformed rewards z_g and errors $\delta_g = z_g - a_g$, the exact advantage difference is

$$\begin{aligned} \widehat{A}_g^{z} - \widehat{A}_g^{\mathrm{audit}} &= \delta_g - \frac{1}{K}\sum_{j\in\mathcal{K}} \delta_j, \\ \left|\widehat{A}_g^{z} - \widehat{A}_g^{\mathrm{audit}}\right| &\le |\delta_g| + \frac{1}{K}\sum_{j\in\mathcal{K}} |\delta_j|. \end{aligned} \tag{14}$$

Both sides use the same retained group. Increasing K alone removes neither the mean error nor the selection effect. A clipped-gradient guarantee would further require control of score-function magnitudes, clipping, and the sampling distribution. These reward identities provide no such guarantee; performance and detector errors require independent measurement.

D GROUP-LEVEL MECHANISM

We compare the centered advantages in Eq. (4) on a fixed sibling group, using the same verifier rewards and frozen detector scores. Holding the group fixed separates the update operator from changes in the policies' rollout distributions.

Residual positive advantage under shaping. Let $z_g = (1-\lambda s_g)v_g$ and $\mu_G = G^{-1}\sum_g z_g$. For $G > 1$ and a verifier-passing trajectory h,

$$A_h^{\mathrm{SHAPE}} > 0 \quad\Longleftrightarrow\quad z_h > \mu_G \quad\Longleftrightarrow\quad z_h > \frac{1}{G-1}\sum_{j\neq h} z_j. \tag{15}$$

Validity of h requires a separate label and cannot be inferred from its advantage. Suppose a group contains n_c correct successes with reward z_c, n_h invalid successes with reward z_h, and n_f verifier failures with reward zero. Then an invalid success has positive SHAPE advantage exactly when

$$n_f z_h > n_c(z_c - z_h). \tag{16}$$

Ordinary failures can therefore leave a discounted invalid success above the group baseline.

Proposition 3 (Kept-group shift between SHAPE and FULL)**.** *Let $\mathcal{K}$ and $\mathcal{R}$ partition the same group into $K > 0$ kept and $M > 0$ removed trajectories, with $G = K + M$. Let μ_K and μ_R be their means under the shaped rewards z_g. Then*

$$A_g^{\mathrm{FULL}} - A_g^{\mathrm{SHAPE}} = \mu_G - \mu_K = \frac{M}{G}(\mu_R - \mu_K) \qquad (g \in \mathcal{K}). \tag{17}$$

For any $i, j \in \mathcal{K}$, the pairwise advantage difference is unchanged.

Proof. The complete-group mean is $\mu_G = (K\mu_K + M\mu_R)/G$. Subtract $z_g - \mu_G$ from $z_g - \mu_K$ to obtain the identity. The common shift cancels when subtracting two kept advantages. □

The mean SHAPE advantage of removed trajectories is $\bar{A}_{\mathcal{R}}^{\mathrm{SHAPE}} = K(\mu_R - \mu_K)/G$. Consequently the common kept shift also equals $(M/K)\bar{A}_{\mathcal{R}}^{\mathrm{SHAPE}}$. A positive mean removed advantage raises the kept advantages. It establishes neither that every removed advantage was positive nor that every removed trajectory was invalid. At $\lambda = 0.6, c = 0.7$, removed verifier successes have residual shaped reward in $[0.4, 0.58)$. Whether this exceeds the kept mean depends on the group.

Worked example: a mixed group. Consider $G = 8$, $\lambda = 0.6$, and $c = 0.7$, with one correct success at score 0.05, two invalid successes at score 0.90, and five verifier failures at score 0.10. Their shaped rewards and advantages are computed in Table 7. The complete-group mean is $\mu_G = 0.23625$. FULL removes the two invalid successes, leaving $K = 6$ and $\mu_K = 0.161666\ldots$. The common kept shift is $0.074583\ldots$. SHAPE still assigns positive advantages to the invalid successes; FULL omits them from the reward surrogate. The correct success's advantage rises, while each failure's negative advantage becomes smaller in magnitude.

Table 7: Centered advantages for the specified mixed group. For removed trajectories, define $A_g^{\mathrm{FULL}} = 0$ to denote their zero reward-surrogate contribution; they are excluded from the kept-group mean. Computed values are rounded for display.

Type (count)	V	s	z	A^{SHAPE}	A^{FULL}
Correct success (1)	1	0.05	0.97	0.73375	0.80833
Invalid success (2)	1	0.90	0.46	0.22375	0
Verifier failure (5)	0	0.10	0	−0.23625	−0.16167

A counterexample to universal improvement. Keep the same scores and rewards but use six correct successes and two invalid successes, with no verifier failures. Now $\mu_G = 0.8425$ and $\mu_K = 0.97$. SHAPE gives every correct success advantage 0.1275 and every invalid success advantage -0.3825. FULL keeps only equal-reward correct successes, whose centered advantages are all 0. Filtering removes the reward signal that distinguished correct from invalid successes in this group; KL may still contribute according to the implementation. Thus filtering does not strengthen the useful group-relative signal in every group.

Why FULL can differ from FILTER. On a fixed group and threshold, FILTER and FULL select the same $\mathcal{K}$. For two kept trajectories i, j with $v_i = v_j = 1$,

$$A_i^{\mathrm{FILTER}} - A_j^{\mathrm{FILTER}} = 0, \qquad A_i^{\mathrm{FULL}} - A_j^{\mathrm{FULL}} = \lambda(s_j - s_i). \tag{18}$$

For example, scores 0.05 and 0.60 both survive $c = 0.7$. Their FILTER rewards are both one, while their FULL rewards are 0.97 and 0.64. The pairwise FULL advantage difference is 0.33, regardless of the other kept rewards. This distinction is useful only if the scores track validity. Both variants keep the same set, so their immediate empty-group rates coincide.

Imperfect detection and retained composition. For a population of verifier successes partitioned into valid successes and invalid successes, let $q \in (0, 1)$ be the invalid proportion. Let r be the removal probability for invalid successes and α the removal probability for valid successes. If the kept population is nonempty, its invalid share is

$$q_{\mathrm{kept}} = \frac{q(1-r)}{q(1-r) + (1-q)(1-\alpha)}. \tag{19}$$

It follows directly that $q_{\mathrm{kept}} < q$ if and only if $r > \alpha$. The identity alone does not guarantee retention of rare correct solutions, informative sibling groups, or enough effective training samples. These determine whether the improved composition is useful.

Conditional removal sensitivity. For a protocol-specific case, set $q = 0.417$ to the reported vanilla-GRPO conditional strict-audit rejection share on SWE-bench Verified (Table 9). Here the two classes are audit-rejected and audit-accepted verifier successes. Let $F = 1$ denote removal, and specify $r = P(F = 1 \mid \mathcal{A} = 0, \mathcal{V} = 1)$ and $\alpha = P(F = 1 \mid \mathcal{A} = 1)$. Table 8 evaluates Eq. (19) at four specified parameter pairs. The retained fraction among verifier successes is $L = q(1-r) + (1-q)(1-\alpha)$; the audit-accepted retention probability is $1 - \alpha$. These quantities describe the filtered population, not the policy's success rate after retraining.

Aggregation, clipping, and KL. The complete parameter update also depends on aggregation, clipping, and KL. Consider $\theta = \theta_{\mathrm{old}}$, an inactive clipping boundary, no KL contribution, and equal weighting of trajectories with token-averaged score functions $u_g = T_g^{-1} \sum_t \nabla_\theta \log \pi_\theta(y_{gt} \mid x_{gt})$. For the convention that divides FULL by K and SHAPE by G, let $d = \mu_G - \mu_K$. The reward-gradient difference for this task is

$$\begin{aligned} g_F - g_S = &\left(\frac{1}{K} - \frac{1}{G}\right) \sum_{g \in \mathcal{K}} A_g^{\mathrm{SHAPE}} u_g \\ &+ \frac{d}{K} \sum_{g \in \mathcal{K}} u_g - \frac{1}{G} \sum_{g \in \mathcal{R}} A_g^{\mathrm{SHAPE}} u_g. \end{aligned} \tag{20}$$

Table 8: Conditional removal sensitivity for $q = 0.417$. Each row specifies class-dependent removal probabilities and computes the retained population's size and strict-audit composition. $1-\alpha$ denotes retention of audit-accepted verifier successes.

Specified removal probabilities (%)		Conditional population quantities (%)		
r	α	L	q_{kept}	$1-\alpha$
60	5	72.065	23.146	95.0
80	5	63.725	13.087	95.0
80	15	57.895	14.405	85.0
95	10	54.555	3.822	90.0

The terms separate reweighting, the baseline shift, and removed contributions. If FULL retains denominator G, the first term disappears and d/K becomes d/G. Different token or task weights alter this expansion. Clipping can activate on later PPO steps, and KL adds a separate gradient. Shared parameters can also change the probabilities of excluded trajectories. The advantage comparison therefore does not bound performance.

Group-level hypotheses. We hypothesize that FULL's additional benefit over SHAPE concentrates in groups containing invalid verifier successes with positive shaped advantages, provided enough correct solutions and informative groups survive. For FULL versus FILTER, we hypothesize that the difference depends on validity information in scores below the threshold. These hypotheses link the group-level learning signals to independent task outcomes.

E Benchmark results by evaluation endpoint

Tables 9–11 report apparent pass@1, conditional protocol-hacking share, and strict-audit pass@1. Entries are percentage means $\pm$ one standard deviation over three seeds. The strict endpoint, P_{strict}, requires verifier success and acceptance by all four audit checks. The conditional share h_{strict} uses verifier-passed attempts as its denominator. Signal (1) only uses the adversarial-test endpoint without the integrity and patch-judge components.

Table 9: SWE-bench Verified performance under the strict protocol. The signal-(1)-only column is the adversarial-test-only endpoint.

Method	Apparent pass	h_{strict}	P_{strict}	Signal (1) only
Vanilla GRPO	52.3 ± 0.4	41.7 ± 0.5	30.5 ± 0.5	31.2 ± 0.5
GRPO + held-out tests	50.1 ± 0.5	33.4 ± 0.6	33.4 ± 0.6	33.9 ± 0.5
GRPO + LLM-judge filter	49.4 ± 0.5	25.6 ± 0.6	36.7 ± 0.6	36.9 ± 0.5
DPO on hack/clean pairs	47.6 ± 0.4	19.4 ± 0.6	38.4 ± 0.5	38.7 ± 0.5
HackFree-GRPO (shape)	51.6 ± 0.4	9.3 ± 0.4	46.8 ± 0.5	47.4 ± 0.5
HackFree-GRPO (filter)	51.0 ± 0.4	7.6 ± 0.4	47.1 ± 0.5	47.8 ± 0.5
HackFree-GRPO (full)	52.0 ± 0.4	5.8 ± 0.3	49.0 ± 0.4	49.9 ± 0.5

F Additional detector and training results

F.1 Additional repository-harness condition

Table 12 gives the results reported as "GRPO + AutoHarness." AutoHarness studies synthesized agent harnesses (Lou et al., 2026). The protection settings behind this label need repository-specific documentation, including any relation to the adversarial-harness condition in Figure 9.

Table 10: EvilGenie/LiveCodeBench performance. The evaluation split is task-disjoint but distribution-matched to the LiveCodeBench/EvilGenie training tasks. No signal-(1)-only column was provided for this benchmark in that table.

Method	Apparent pass	h_{strict}	P_{strict}
Vanilla GRPO	47.0 ± 0.4	39.2 ± 0.6	28.6 ± 0.5
GRPO + held-out tests	44.6 ± 0.5	31.0 ± 0.7	30.8 ± 0.6
GRPO + LLM-judge filter	43.3 ± 0.5	23.1 ± 0.7	33.3 ± 0.6
DPO on hack/clean pairs	42.0 ± 0.5	18.0 ± 0.6	34.4 ± 0.6
HackFree-GRPO (shape)	46.2 ± 0.5	8.7 ± 0.4	42.2 ± 0.5
HackFree-GRPO (filter)	45.8 ± 0.5	7.1 ± 0.4	42.5 ± 0.5
HackFree-GRPO (full)	46.7 ± 0.5	5.4 ± 0.4	44.2 ± 0.4

Table 11: Held-out-repository performance under the strict audit. Entries summarize the same reported seeds.

Method	Apparent pass	h_{strict}	P_{strict}
Vanilla GRPO	38.4 ± 0.6	35.1 ± 0.6	24.9 ± 0.6
GRPO + held-out tests	37.1 ± 0.6	27.6 ± 0.6	26.9 ± 0.5
GRPO + LLM-judge filter	36.5 ± 0.6	21.0 ± 0.7	28.8 ± 0.6
DPO on hack/clean pairs	35.8 ± 0.6	17.1 ± 0.6	29.7 ± 0.6
HackFree-GRPO (shape)	39.5 ± 0.5	8.0 ± 0.4	36.3 ± 0.5
HackFree-GRPO (filter)	39.2 ± 0.5	6.5 ± 0.4	36.7 ± 0.5
HackFree-GRPO (full)	40.1 ± 0.5	4.9 ± 0.4	38.1 ± 0.5

Table 12: The reported AutoHarness-labeled condition (%, mean $\pm$ seed standard deviation). Its SWE-bench Verified signal-(1)-only result is 38.4 ± 0.5. No equality of its configuration with the separate adversarial-harness condition is assumed.

Benchmark	Apparent pass	h_{strict}	P_{strict}
SWE-bench Verified	48.8 ± 0.5	22.1 ± 0.6	38.0 ± 0.6
EvilGenie / LCB	42.9 ± 0.5	19.8 ± 0.7	34.4 ± 0.6
Held-out repositories	36.2 ± 0.7	18.5 ± 0.6	29.5 ± 0.5

F.2 SOURCE VALUES FOR THE CROSS-BENCHMARK COMPARISON

Table 13 gives the cross-benchmark panel's values. All rates use the strict audit-pass endpoint P_{strict} defined in Section 4.

Table 13: Cross-benchmark strict audit-pass@1 (%). Entries are point estimates; gains are in percentage points.

Benchmark	Vanilla GRPO	HackFree-GRPO	Gain (pp)
SWE-BENCH LIVE	39.6	46.1	+6.5
Held-out repos	24.9	38.1	+13.2
COUNTDOWN-CODE	51.0	55.7	+4.7
EvilGenie unseen	42.0	47.2	+5.2

G TRAINING CONFIGURATION

Table 14 lists the reported policy-training configuration. Collection rounds, optimizer updates, and reused minibatches are counted separately. The rollout budget includes generated trajectories that are subsequently filtered.

Table 14: Policy-training and detector reference settings.

Item	Setting
Primary policy	Qwen2.5-Coder-7B-Instruct
Training task pools	SWE-Gym: 2,438; LCB-EvilGenie: 812
Evaluation task pools	SWE-bench Verified: 500; held-out repositories: 200
Seeds	$\{11, 23, 47\}$
Group / collection size	$G = 8$; 256 trajectories per collection
Generated trajectory budget	24,000
Optimizer updates	4,000; extended run to 8,000
Maximum tool turns	50 per trajectory
LoRA rank / alpha	64 / 128 (Hu et al., 2022)
Learning rate / warmup	5×10^{-6} / 100 optimizer steps
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$
Shaping / filtering	$\lambda = 0.6$, $c = 0.7$
KL coefficient / clipping	$\beta = 0.04$, $\epsilon = 0.2$
Policy-training node	8×H100 80GB
Infrastructure	SkyRL-Agent and SWE-MiniSandbox (Cao et al., 2025; Yuan et al., 2026)
Detector backbone	DeBERTa-v3 base with three input streams
Stream cap	512 tokens per stream; 1,536 before fusion
Seed labels	210; split 150/30/30
Offline self-training rounds	3
Pseudo-label cutoffs	$s \leq 0.05$ or $s \geq 0.95$
Detector construction pool	8,512; class counts 4,236 and 4,276

Resource accounting. Resource accounting tracks elapsed hours, accelerator-hours, CPU work, and API charges. With a constant allocation, accelerator-hours equal elapsed hours multiplied by the allocated accelerator count; total cost sums over executed runs. Shared preprocessing and reused checkpoints are counted once. Online overhead includes the generation of discarded trajectories and excludes evaluation-only audits.

H ADDITIONAL BEHAVIORAL AND VERIFIER ANALYSES

The panels report protocol-based outcomes. Axes labelled "true pass" measure strict-audit success.

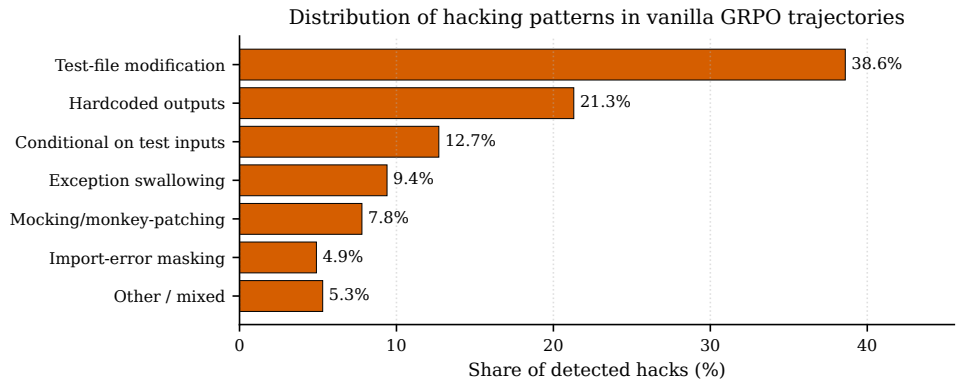

Figure 5: Distribution of protocol-defined hacking patterns. Test-file modification includes violations of the strict benchmark rule, including edits that can be benign in ordinary maintenance.



Figure 6: Detector and audit failure modes. Reflection-based monkey-patching can evade observable diff features; benign-looking fixture changes can violate the strict protocol; contract relaxation can create borderline judgments. Frequencies describe these categories in the reported audit, not a complete confusion matrix.

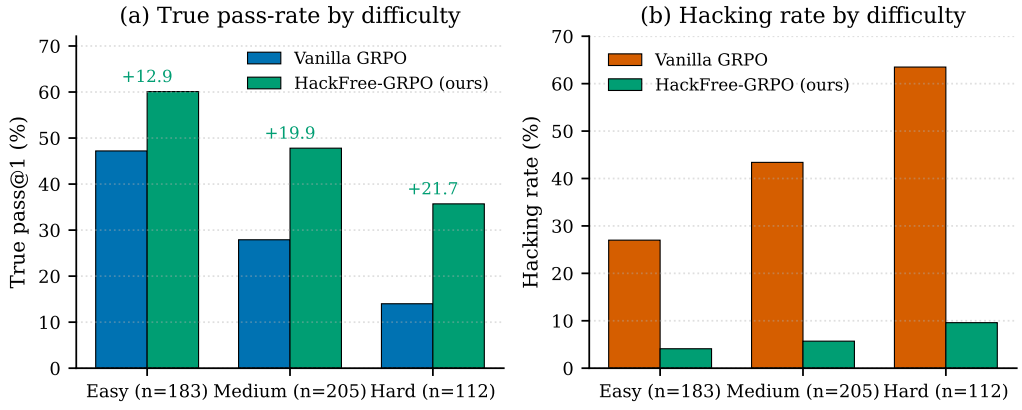

Figure 7: Difficulty-stratified audit results, with task counts shown in the labels. Conditional hacking rates require the verifier-passed count within each bucket; they are not aggregated by task counts alone. These panels are presented as the reported subgroup summaries.

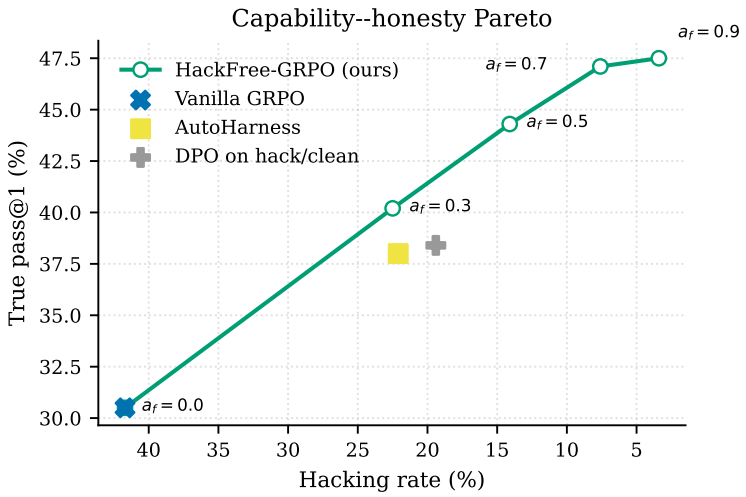

Figure 8: Audit-success and conditional-hacking tradeoff along the reported sweep. The panel's a_f indexing is separate from the threshold symbol c used in the update equations; this panel is not used to select c.

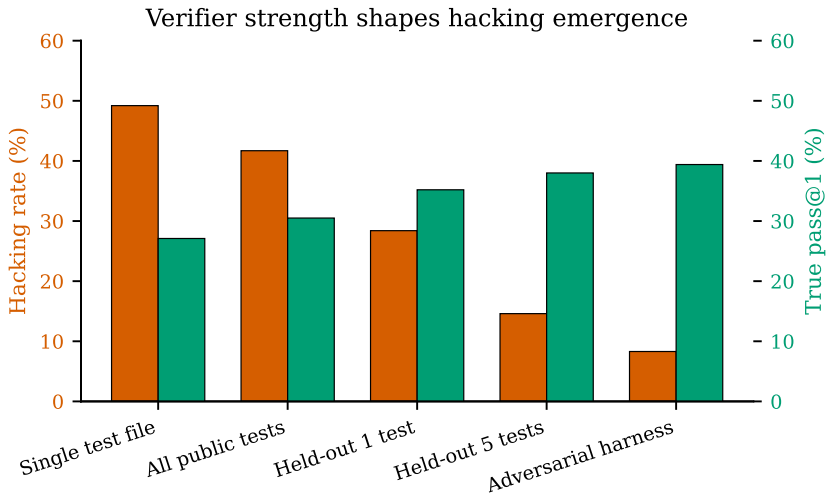

Figure 9: Outcomes under five verifier configurations. The plotted adversarial-harness condition is reported separately; no one-to-one equivalence with the AutoHarness-labeled training baseline is assumed. Both axes use protocol-defined endpoints.

I METRIC INTERPRETATION AND PAIRED UPDATE ANALYSIS

I.1 AUDIT ENDPOINTS AND THEIR DENOMINATORS

Strict audit-pass performance requires verifier acceptance and all declared execution, infrastructure-integrity, and patch-judgment checks; h_{strict} is conditional on verifier acceptance. Its numerator and denominator use the same evaluation frame. Revising the treatment of test refactors or ambiguous judgments defines a different audit and requires rescoring the outputs.

Official-harness success O, human semantic correctness C, and human confirmation of explicit exploitation B measure different properties. Evidence of exploitation beyond an audit rejection or an incorrect repair is required to establish $B = 1$. Legitimate test maintenance can also trigger a protocol rejection. Tables 1 and 2 use the stated strict and signal-(1)-only endpoints. Table 3 gives scenario references and prospective independent-outcome targets under the definitions in Appendix J.

An audit of verifier-passed outputs estimates a conditional rate. For a stratified sample, let N^+ denote the number of passed outputs, N_h the population size of stratum h, and n_h its audited size. With uniform sampling within strata,

$$\widehat{P}(C = 1 \mid V = 1) = \frac{1}{N^+} \sum_h \frac{N_h}{n_h} \sum_{j \in \mathcal{S}_h} \mathbf{1}[C_j = 1]. \tag{21}$$

Multiplying this estimate by $P(V = 1)$ yields the joint quantity $P(V = 1, C = 1)$, not unconditional $P(C = 1)$. The same distinction applies to exploitation labels. Assigning unresolved judgments to either binary outcome gives sensitivity bounds while preserving their unresolved status.

Paired comparisons distinguish task-cluster uncertainty from variation across training seeds. Differences of reported means are directly computable, but their paired variances also require covariance between methods. The arithmetic contrasts in Table 2 therefore report mean differences only.

I.2 GROUP COMPOSITION AND LOCAL SIGNAL ALLOCATION

To isolate the update operator, paired analysis applies every rule to the same complete pre-update sibling group, fixing its source policy, verifier outcomes, and detector scores. Group-composition labels must be independent of those scores. Comparisons between different on-policy rollout distributions also reflect changes in the sampled trajectories.

Let $H_i = \sum_g \mathbf{1}[V_{ig} = 1, B_{ig} = 1]$ count rewarded exploits and $Q_i = \sum_g \mathbf{1}[V_{ig} = 1, C_{ig} = 1, B_{ig} = 0]$ count correct, legitimate passes. A mixed group has $H_i > 0$ and $Q_i > 0$. Its contamination share is $q_i = H_i / \sum_g \mathbf{1}[V_{ig} = 1]$; H_i/G describes contamination relative to all generated candidates. Clean groups require independent confirmation of their passing candidates. Detector flags alone cannot establish this category; no-pass groups and unresolved cases are recorded separately.

For a label class L, define positive advantage weight

$$W_L^+ = \sum_g w_g [\widehat{A}_g]_+ \mathbf{1}[g \in L], \tag{22}$$

where w_g is the trajectory weight used by the update. This quantity measures positive advantage allocation to each label class, distinguishing reduced weight on confirmed exploits from increased weight on correct solutions. Computing a gradient norm additionally requires trajectory-specific score-function gradients; final task performance requires evaluation of the updated policy.

Removal, recentering, and renormalization. For a fixed nonempty retained group, let $r_g = (1 - \lambda s_g) V_g$, $\mu = G^{-1} \sum_g r_g$, and $\mu_K = K^{-1} \sum_{g \in \mathcal{K}} r_g$. Under mean-normalized trajectory weighting, let ψ_g denote the policy-generated-token-averaged log-probability gradient at the rollout

parameters. With scores and advantages held fixed, the local reward-surrogate gradient difference is

$$\begin{aligned} g_{\mathrm{FULL}} - g_{\mathrm{SHAPE}} = & -\frac{1}{G}\sum_{g\notin\mathcal{K}}\psi_g(r_g-\mu) && \text{(removal)} \\ & +\frac{\mu-\mu_K}{G}\sum_{g\in\mathcal{K}}\psi_g && \text{(recentering)} \\ & +\left(\frac{1}{K}-\frac{1}{G}\right)\sum_{g\in\mathcal{K}}\psi_g(r_g-\mu_K) && \text{(renormalization).} \end{aligned} \tag{23}$$

This local identity holds at the rollout parameters, where PPO ratios equal one, with KL contributions excluded. Other loss normalizations change the weighting terms. Extending the identity to complete training dynamics or a numerical performance gain requires further analysis.

FULL and FILTER share the same retained set, so under the same weighting their local reward-gradient difference is

$$g_{\mathrm{FULL}} - g_{\mathrm{FILTER}} = -\frac{\lambda}{K}\sum_{g\in\mathcal{K}}\psi_g\left(V_g s_g - \overline{Vs}_{\mathcal{K}}\right). \tag{24}$$

When all retained verifier rewards equal one, FILTER assigns zero centered advantages and FULL can distinguish candidates through their scores. Alignment with correctness depends on score quality, as the worked examples and counterexample in Appendix D illustrate.

A score-specific counterfactual control. Permuting scores separately among verifier-passed and verifier-failed candidates within a group preserves each stratum's score multiset, retained size and verifier-outcome counts, transformed-reward distribution, baseline, and normalization. By breaking the score–trajectory association, this operator-level control isolates label-specific reweighting from changes in group size or reward distribution. It defines a counterfactual comparison without adding an empirical result.

Empty and uninformative groups. At fixed scores and threshold, FILTER and FULL have identical K and empty-group indicators. Singletons and groups with identical transformed rewards have zero centered advantages. Accounting therefore distinguishes generated candidates, unique pre-filter groups, empty groups, and informative updates. Resampling an empty group consumes additional budget. Any difference in empty-group rates between later on-policy runs reflects their changed rollout distributions.

I.3 Filter precision and correct-candidate retention

For removal indicator F, independent labels define

$$\mathrm{Prec}_{\mathrm{filter}} = P(B=1 \mid F=1, V=1), \qquad R_{\mathrm{clean}} = P(F=0 \mid C=1, B=0, V=1). \tag{25}$$

False-suppression incidence is instead $P(F=1, C=1, B=0 \mid V=1)$. Each quantity uses a different denominator: precision among filtered candidates and retention of correct candidates must be assessed separately. Their interpretation requires independent labels, explicit handling of uncertainty, and a sampling frame defined before filtering. Aggregate pass rates and validation F1 leave these quantities undetermined.

I.4 Scope of transfer and adaptive behavior

The cross-benchmark evidence covers task transfer under the evaluated policy and audit protocols; transfer to another policy family requires separate evidence. Extended ordinary training characterizes the tested training dynamics. Assessing adaptive evasion additionally requires an attack protocol and independent exploitation labels: a confirmed evasion must pass the verifier and the detector while exhibiting independently established exploitation.

J OUTCOME REFERENCES, PROSPECTIVE TARGETS, AND SENSITIVITY

Empirical anchors and identification. For each method, take the reported rates $a = P(A = 1)$, $t = P(T = 1)$, and $v = P(V = 1)$ from Tables 1 and 9. Under the scenario's nesting assumption $A \subseteq T \subseteq V$, they define four masses: a, $t-a$, $v-t$, and $1-v$. An independent outcome Y satisfies

$$P(Y = 1) = au_1 + (t - a)u_2 + (v - t)u_3 + (1 - v)u_4,$$

where u_j is its conditional success probability in stratum j. The reported rates leave these probabilities unidentified, so each projection requires an explicit assumption about the outcomes within each stratum.

Central alignment scenario. The central official-execution scenario sets $O = T$. For human correctness, it treats all A successes as correct, assigns a fraction $\omega = 1/2$ of $T \setminus A$ to correct repairs, and assigns no correct cases to $V \setminus T$. For confirmed exploitation it assigns a fraction $\kappa = 1/2$ of $V \setminus T$. Thus

$$\widehat{P}(O = 1) = t, \tag{26}$$

$$\widehat{P}(C = 1 \mid V = 1) = \frac{a + \omega(t - a)}{v}, \tag{27}$$

$$\widehat{P}(B = 1 \mid V = 1) = \frac{\kappa(v - t)}{v}. \tag{28}$$

This midpoint allocation has not been calibrated against human labels. It makes C and B mutually exclusive but allows cases in neither category. The general outcome definitions allow correct-but-exploiting trajectories, which this particular scenario excludes.

Targets retaining ninety percent of the reference improvement. Let $\mathbf{u}_m$ collect the scenario reference rates for official success, conditional human correctness, and conditional exploitation for method m. We set the prospective FULL target vector to

$$\mathbf{q}_{\text{FULL}} = \mathbf{u}_{\text{FILTER}} + \rho\big(\mathbf{u}_{\text{FULL}} - \mathbf{u}_{\text{FILTER}}\big), \qquad \rho = 0.9. \tag{29}$$

Using the unrounded reference rates gives targets of 49.69% for official success, 94.89% for conditional human correctness, and 2.13% for conditional exploitation. The first two are minimum targets and the last is a maximum target. Displayed values are rounded; Eq. (29) defines the unrounded thresholds. For official success, the calculation is $47.80 + 0.9(49.90 - 47.80) = 49.69$. The reference values are specified comparison points, not identified ideal performance bounds. Equation (29) defines experimental targets; it does not alter the reference outcome model or establish that a trained policy attains them. Measurements of both FULL and its controls, with paired uncertainty, are required to assess the comparison. This target rule applies to independent outcome rates; group-level expectations are determined by their specified error model.

Literature context. EvilGenie reports seven explicit-hacking examples among fourteen visible-pass/holdout-fail outputs; the remaining cases include heuristics and ordinary solution errors (Gabor et al., 2025). This motivates the half-allocation scenario, although the small sample and different tasks prevent direct calibration. SWE-ABS rejects 2,184 of 11,041 previously passing patches under strengthened tests (Yu et al., 2026), illustrating how much execution endpoints can differ. These findings provide no common correction factor across methods. ARA reports a code Pass@1 change from 34.2 to 35.8, and GRIFT reports Code True Accuracy from 19.6 at the starting checkpoint to 23.3 after its intervention (Beigi et al., 2026; Wang et al., 2026a). Because their tasks, models, and update procedures differ, neither gain enters the HackFree estimates.

Endpoint sensitivity. Let $f = P(O = 0 \mid T = 1)$ and $g = P(O = 1 \mid T = 0)$. Then

$$\widehat{P}(O = 1) = t(1 - f) + (1 - t)g.$$

Table 15 compares the central alignment scenario with two specified five-percent disagreement scenarios. The error rates are chosen for this analysis. Shared rates can preserve the method ordering mechanically, while method-specific disagreement could change it. The last row applies Eq. (29) to the FULL and FILTER references under each common mapping. Because the mapping is affine in the reference success rate, this equals applying it to the central target. These transformed targets are scenario comparisons, not uncertainty intervals.

Table 15: Official-outcome references and target sensitivity (%). The first four rows are scenario references; the last gives minimum targets retaining 90% of FULL's improvement over FILTER within each mapping. These scenarios describe mapping sensitivity, not statistical uncertainty.

Method	$(f, g) = (.05, 0)$	$(f, g) = (0, 0)$	$(f, g) = (0, .05)$
Vanilla GRPO	29.64	31.20	34.64
SHAPE	45.03	47.40	50.03
FILTER	45.41	47.80	50.41
FULL reference	47.41	49.90	52.41
FULL (90% target)	≥47.21	≥49.69	≥52.21

Group-error model. Table 4 uses fixed group compositions. Each invalid success independently receives score .90 with probability $r = .8$ and .60 otherwise. Each valid success receives score .90 with probability $\alpha = .05$ and .05 otherwise. Verifier failures have score .10 and reward zero. We specify these detection rates and scores as sensitivity conditions; they are not measurements of HackDetect. The model includes both missed detections and false rejections, motivated by errors and domain shifts documented in related detection studies (Gabor et al., 2025; Beigi et al., 2026; Wang et al., 2026a).

For every error assignment, compute SHAPE and FULL on the same group, define

$$W_L^+ = \frac{1}{G} \sum_{g \in L} [A_g]_+,$$

and average over the exact product-Bernoulli probabilities. Both operators use denominator G to isolate removal and recentering. The columns are $E[W_H^{+,\mathrm{SHAPE}} - W_H^{+,\mathrm{FULL}}]$, $E[W_C^{+,\mathrm{FULL}} - W_C^{+,\mathrm{SHAPE}}]$, and the expected retained valid count. The first two columns show advantage weights multiplied by 100 for legibility; these are not pass-rate percentage points. A loss using $1/K$, token weighting, clipping, or KL requires a corresponding update calculation. Relating these examples to final accuracy also requires the frequency of each group composition during training.

Interpretation. The references describe the specified outcome assumptions, and the targets express a prospective comparison criterion. Official leaderboard comparisons and state-of-the-art claims require measured outcomes on the same task population, with a common evaluation procedure and resource budget.

K Extended related work

Proxy objectives and the meaning of reward hacking. Reward misspecification and specification gaming are established concerns in AI safety (Amodei et al., 2016; Krakovna et al., 2020); Skalse et al. (2022) study formal relationships between proxy and intended rewards. An analysis of finite evaluation considers reward hacking under an explicit set of modeling assumptions (Wang & Huang, 2026). These analyses distinguish the verifier from the task objective. Failure on an additional test alone leaves open whether a repair is incomplete or exploits the verifier. Recent RLVR work studies shortcuts in reasoning (Helff et al., 2026), programming-agent reward hacking (Gabor et al., 2025), and its emergence and generalization in a controlled code testbed (Khalifa et al., 2026). Taylor et al. (2025) study transfer after supervised fine-tuning on reward-hacking examples. Their transfer results concern this targeted exposure; whether ordinary repository-level RLVR produces the same deployment behavior remains open.

Detection and interventions during learning. Adversarial Reward Auditing combines adversarial discovery with an auditor and uses auditor-guided reward gating during RLHF (Beigi et al., 2026). Gradient Fingerprints use internal gradient representations for detection and also study rejection fine-tuning as a mitigation (Wang et al., 2026a). TRACe investigates detection through reasoning effort (Wang et al., 2026b). These methods use different detection signals and intervene at different stages

of training. Work on chain-of-thought faithfulness cautions against taking a model's verbal reasoning as a faithful account of its decision process (Chen et al., 2025). Composite verifiable rewards have also been studied in medical question answering, combining answer correctness with targeted penalties for answer-disclosure and structural violations (Tarek & Beheshti, 2025). Our detector consumes repository actions, diffs, and execution logs. Evaluating it requires independent task outcomes and an account of the valid repairs suppressed by filtering.

Policy optimization and preference learning. Learning from human preferences provides one route to instruction-following policies (Ouyang et al., 2022); DPO offers a direct preference-learning objective (Rafailov et al., 2023). PPO supplies the clipped-surrogate lineage (Schulman et al., 2017), while DeepSeekMath and DeepSeek-R1 develop group-relative or verifiable-reward reasoning training (Shao et al., 2024; Guo et al., 2025). DAPO modifies several parts of the training recipe (Yu et al., 2025). Their relative benefits depend on which parts of the learning process change: feedback, optimization, or sampling. Testing one component answers a narrower question than comparing complete training recipes.

Execution feedback and code-agent training systems. RLEF studies reinforcement learning with execution feedback for code models (Gehring et al., 2025), and SWE-Gym supplies executable tasks for training software-engineering agents and verifiers (Pan et al., 2025). SkyRL-Agent addresses multi-turn agent RL infrastructure (Cao et al., 2025), while SWE-MiniSandbox studies container-free RL for software-engineering agents (Yuan et al., 2026). Related training approaches include SWE-Fixer for issue resolution (Xie et al., 2025), subtask-oriented reinforced fine-tuning in SoRFT (Ma et al., 2025a), and multi-turn GRPO for self-correcting code generation in MURPHY (Ekbote et al., 2025). SWE-RM investigates execution-free feedback (Shum et al., 2025), and practical guidance examines multi-turn agentic RL (Wang & Ammanabrolu, 2025). Comparisons across these systems require specifying verifier permissions and measuring hacking within each environment.

Agent interfaces and repository-level workflows. The broader autonomous-agent literature places tool use and interaction with an environment alongside language generation (Wang et al., 2024). SWE-agent focuses on agent-computer interfaces for software engineering (Yang et al., 2024); OpenHands provides an open platform for software development agents (Wang et al., 2025a). CodeAgent examines tool-integrated repository-level coding (Zhang et al., 2024a), and AutoCodeRover studies autonomous program improvement (Zhang et al., 2024b). The threat model depends on access to tests, executable processes, and writable paths; prevalence estimates apply to the corresponding interface.

Benchmarks and evaluation populations. SWE-bench evaluates resolution of real repository issues (Jimenez et al., 2024), and SWE-bench Verified provides a human-filtered subset (SWE-bench Team, 2024). Related evaluation settings include LiveCodeBench for code evaluation (Jain et al., 2025), REPOCOD for repository-level coding (Liang et al., 2025), and DebugBench for debugging (Tian et al., 2024). SWE-bench Live addresses continuing benchmark collection (Zhang et al., 2025); SWE-rebench and SWE-rebench V2 study automated and language-agnostic task collection (Badertdinov et al., 2025; 2026). Task provenance, overlap, evaluation procedures, and the meaning of success must be specified for each cross-benchmark comparison.

Patch correctness and evaluation robustness. Program-repair research distinguishes test-passing plausibility from correctness (Qi et al., 2015), including detection of test-suite-overfitted patches through generated tests (Xin & Reiss, 2017). SWE-bench correctness audits revisit the interpretation of solved issues (Wang et al., 2025b), and SWE-ABS studies adversarial benchmark strengthening (Yu et al., 2026). Dynamic unit-test scaling examines tests as a code reward-modeling resource (Ma et al., 2025b). A failed strengthened test exposes inadequate behavior under that test; identifying deliberate verifier manipulation requires additional behavioral evidence. Broader adversarial-ML terminology also emphasizes explicit attack and mitigation definitions (Vassilev et al., 2025).

The scope of synthesized harnesses. AutoHarness synthesizes code harnesses for agents and studies action legality and related code-based support in TextArena environments (Lou et al., 2026). Ap-

plying it to SWE-bench requires a repository-specific implementation, permissions manifest, and validation. Protection against test editing or monkey-patching must be established by that implementation.

Model and adaptation components. The reference configuration uses the Qwen2.5-Coder-7B-Instruct policy (Qwen Team, 2024), LoRA adaptation (Hu et al., 2022), and a detector built on DeBERTa-v3 (He et al., 2023). These references document the components used in our configuration.