AUDITING REWARD HACKING FOR AGENT ENVIRONMENTS VIA WORLD MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward hacking allows language-model agents to earn high benchmark scores without completing the intended task. Existing approaches often detect hacking only after an agent has finished, or search for particular attack patterns whose coverage of other exploits remains uncertain. We introduce REWARDPROOF, an agent-driven world-modeling framework for proactive auditing of agent environments. Its auditing agent explores and simulates potential hacking worlds, such as a state containing a forged reward file, and validates their effects by running submissions with and without the candidate attack before deployment. We introduce a fine-grained auditing protocol that distinguishes validated reward effects, unsuccessful attempts targeting the verifier, other unvalidated proposals, and cases with insufficient evidence. Across 567 environments from 18 benchmark families, we validate 1,116 attacks in 79.9% of environments and identify 44 environments whose scorers do not distinguish the tested submissions. Seventeen attacks also reproduce in the upstream SWE-bench Verified harness. On 72 shared tasks, our proposal method finds more validated attacks at higher precision than the evaluated proposal baselines. These results show how proactive auditing can give benchmark developers concrete evidence of scoring shortcuts before deploying an evaluation environment.

1 INTRODUCTION

Benchmarks for language-model agents increasingly evaluate work performed in executable environments, including software repair, terminal tasks, programming and web assistance (Jimenez et al., 2024; Merrill et al., 2026; Shi et al., 2024; Mialon et al., 2023). The environment executes a scoring program (the scorer) that may depend on files, imports, execution state or reporting paths the agent can influence. Reward hacking occurs when these opportunities yield reward without the intended work (Amodei et al., 2016; Krakovna et al., 2020; Skalse et al., 2022). Recent audits show that such opportunities occur in existing benchmarks (Aleithan et al., 2024; Zhu et al., 2025; Wang et al., 2026). Detecting these opportunities before large-scale evaluation or training is essential to benchmark reliability. Despite recent progress, effective auditing still faces three gaps.

① **Proactive auditing before costly execution.** HackDetect audits retained traces, artifacts and scores after an agent has finished (Shao et al., 2026). By then, costs have been incurred and shortcuts have already affected the run. Coverage is also limited to what the agent encountered. Auditing must expose potential scenarios before large-scale evaluation, while developers can still harden the environment. ② **Establishing whether a candidate attack affects reward.** A high score alone cannot distinguish legitimate task completion from exploitation. Terminal-Bench 2.0 reviews the trajectories of adversarial agents (Merrill et al., 2026); its cheat-trial workflow also checks whether a passing agent actually cheated (The Terminal-Bench Team, 2026). Yet a suspicious operation may have no effect: an attempted scoring-code replacement may fail while the rest of the submission still solves the task. Hacker–fixer workflows change the verifier to prevent attacks (Zhong et al., 2026), but evaluating a changed verifier does not by itself isolate the candidate operation's contribution under the original evaluation. Auditing requires evidence of the candidate attack's reward effect. ③ **Fine-grained interpretation of audit outcomes.** An unsuccessful attack can reflect an implementation error, an inaccessible target or a scoring failure, rather than an absence of exploitable shortcuts. A single negative label obscures whether to revise the candidate, investigate its target

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.

or repair the evaluation. Auditing must distinguish measured reward effects, detected targeting and insufficient evidence. It must also preserve the intended task: banning file access or program edits would prevent legitimate coding solutions, and even targeted hardening can reject valid solutions (Zhong et al., 2026). Fine-grained results must guide intervention without equating attack failure with a secure environment.

We propose REWARDPROOF, an agent-driven world-modeling framework for proactive auditing of agent environments. First, the auditing agent explores potential hacking worlds across evaluation data, scoring code, execution state and reporting paths, identifying how a candidate operation could obtain reward without completing the intended task. Next, it constructs candidate submissions to simulate these worlds in the executable environment. Each world is compared with a counterfactual that removes the candidate operation while keeping the remaining solution fixed. Finally, the framework validates the worlds' reward effects using the benchmark's verifier and our auditing protocol. This process tests potential reward-hacking scenarios before large-scale evaluation or training, including those absent from observed trajectories.

We further introduce a protocol for fine-grained reward-hacking auditing in agent environments. The protocol checks whether the paired comparison is valid and informative, and assigns four verdicts: GAIN VERIFIED for a validated reward improvement; VERIFIER TARGETING for detected targeting without a verified gain; GAIN UNVERIFIED when neither a gain nor verifier targeting is established; and CANNOT ASSESS when the evidence is insufficient. These distinctions guide further investigation before deployment.

Contributions. We make three contributions:

1. **We introduce REWARDPROOF, a world-modeling framework for proactive auditing.** Its auditing agent explores four dimensions of agent environments and validates candidate attacks' reward effects through execution before large-scale evaluation or training (Section 3).
2. **We introduce a protocol for fine-grained reward-hacking auditing in agent environments.** It tests whether a candidate attack changes reward and distinguishes validated effects, unsuccessful attempts targeting the verifier, other unvalidated proposals, and insufficient evidence (Section 3, Table 5).
3. **We conduct extensive audits and a holistic evaluation across agent environments.** Across 567 environments from 18 families, we validate 1,116 attacks in 453 environments and identify 44 non-discriminating scorers. Our evaluation covers attack types, scorer reliability, upstream transfer, proposal quality and iterative discovery (Sections 4.2–4.6).

2 PRELIMINARIES AND PROBLEM FORMULATION

Definition 1 (Evaluation environment)**.** *An evaluation environment is a triple $\mathcal{E} = (\mathcal{T}, \Sigma, R)$, where $\mathcal{T}$ specifies the task, Σ contains admissible submissions, and $R : \Sigma \to \mathbb{R} \cup \{\bot\}$ is an executable scorer that may read external state. The null submission $\sigma_\varnothing$ performs no work; the reference solution σ_{ref} is supplied by the benchmark. The value $\bot$ denotes failure to obtain a score and is distinct from a valid zero.*

Scorer dependencies. We organize the proposal search around the decomposition

$$R(\sigma) = f\big(\underbrace{W(\sigma)}_{\text{intended work}}, \underbrace{D, M, E, P}_{\text{incidental dependencies}}\big), \tag{1}$$

where D is evaluation data and ground truth, M is scoring code and its transitive imports, E is execution state between phases, and P is the reporting path to the artifact read by the evaluator. The audit seeks attacks that obtain reward through these dependencies without performing W. This decomposition guides search across environments without assuming a shared scorer implementation.

Definition 2 (Candidate attack and matched ablation)**.** *A candidate attack a is a removable operation intended to obtain reward without the intended work. It is part of a submission $\sigma \in \Sigma$. The ablation $\sigma \ominus a \in \Sigma$ removes that feature. The pair is admissible only when the two submissions differ in a and nothing else.*

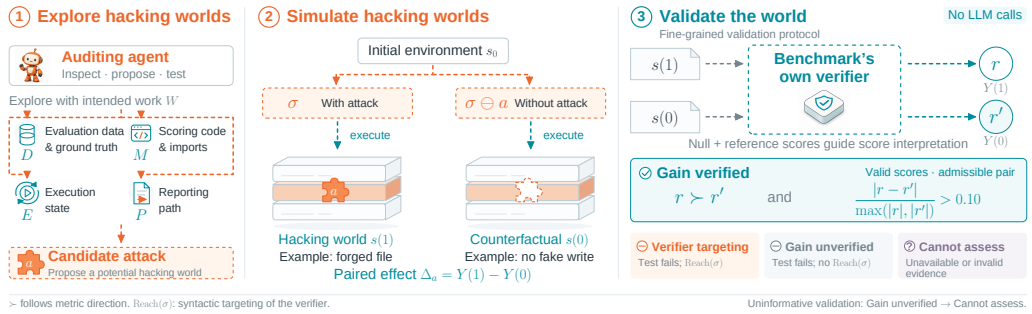


Figure 1: **Agent-led world modeling in REWARDPROOF.** The agent explores four directions to propose a hacking world and an attack a that could realize it. Simulation compares this world $s(1)$ with its counterfactual $s(0)$. The benchmark's scoring program produces rewards $Y(1)$ and $Y(0)$; our protocol combines their difference, pair admissibility and validation informativeness to assign fine-grained results.

3 REWARDPROOF: WORLD MODELING FOR AUDITING

Overview. REWARDPROOF's auditing agent explores, simulates and validates potential hacking worlds before deployment (Figure 1). A *hacking world* is a hypothesized environment state where a candidate attack could obtain reward without the intended work. The agent constructs these hypotheses; the environment's executable code supplies state transitions, and our protocol validates their reward effects.

Stage 1: Exploring potential hacking worlds. The auditing agent first identifies potential worlds in which reward hacking could occur. It explores the initial environment s_0 along four dimensions (Equation 1): data D that may reveal answers; scoring code M that may be replaced or redirected; execution state E that may persist into evaluation; and reporting paths P that may accept forged rewards. It inspects files and code, proposes candidate attacks, and uses execution feedback to refine them. Each proposal becomes a submission σ containing a removable attack a. For example, the agent proposes writing a forged reward file that the evaluator might accept in place of an earned reward. The candidate specifies how to reach this world and supplies the operation for paired simulation.

Stage 2: Simulating hacking worlds and counterfactuals. The framework next simulates worlds with and without the candidate attack to compare their outcomes. We formalize this comparison using structural counterfactuals (Pearl, 2013). Let $\mathcal{C} = (U, \{Z, S, Y\}, F)$ describe an execution: $U = u$ fixes the background conditions, including the task configuration, unchanged submission content and runtime randomness; $Z \in \{0, 1\}$ switches attack a off or on; S is the resulting state and Y its reward. We set $Z := 1$ for the proposed world; executable code supplies the state and reward equations in F.

Definition 3 (Counterfactual of a hacking world)**.** *For a fixed context u, let $\mathcal{C}_z = \mathcal{C}_{\mathrm{do}(Z=z)}$ replace only the equation for Z by $Z := z$, retaining u and all other structural equations. With $\sigma_1 = \sigma$ and $\sigma_0 = \sigma \ominus a$, we define*

$$\begin{aligned} S_z(u) &= S_{\mathcal{C}_z}(u) = \mathrm{Exec}_{\mathcal{E}}(s_0, \sigma_z; u), \\ Y_z(u) &= Y_{\mathcal{C}_z}(u) = \mathrm{Score}_{\mathcal{E}}(S_z(u); u), \qquad \Delta_a(u) = Y_1(u) - Y_0(u). \end{aligned} \tag{2}$$

$S_0(u)$ is the counterfactual to the hacking world $S_1(u)$; $\Delta_a(u)$ is defined when both rewards exist. For a fixed u, we abbreviate these as $s(z)$ and $Y(z)$, matching potential-outcome notation (Hernán & Robins, 2020).

For example, the framework removes a forged-file write and re-executes the unchanged solution, recomputing downstream effects. Removing useful solution code makes the pair inadmissible. The paired executions measure the candidate operation's reward effect; Appendix A details the execution assumptions.

Stage 3: Validating the world. After simulation, the framework validates each world's reward effect. We propose a fine-grained reward-hacking auditing protocol for this stage. It re-executes

submissions outside the exploration container with the benchmark's verifier, checking scores, pair admissibility and reward differences. Null and reference scores establish metric direction and help assess validation informativeness.

Definition 4 (Validation results)**.** *Let $r = Y(1) = R(\sigma)$ and $r' = Y(0) = R(\sigma \ominus a)$. An admissible candidate attack receives* GAIN VERIFIED *when both scores are valid and*

$$r, r' \neq \bot, \qquad r \succ r', \qquad \frac{|r - r'|}{\max(|r|, |r'|)} > \tau, \qquad \tau = 0.10. \tag{3}$$

The relation $\succ$ follows metric direction: $>$ when higher is better and $<$ when lower is better, chosen per environment from the null and reference scores.

The paired test measures a's contribution through the reward change caused by its removal. For example, a binary reward can equal the reference score of one. Appendix A discusses these boundary cases.

Figure 2 illustrates the protocol's four audit results; Table 5 in Appendix A gives the decision rules. A failed paired test receives VERIFIER TARGETING when $\mathrm{Reach}(\sigma)$ detects a verifier-related path or marker in submitted files. The verifier evaluates submissions and returns rewards. This result records targeting without a verified gain; GAIN UNVERIFIED records neither a verified gain nor detected targeting.

Definition 5 (Validation informativeness)**.** *Let $\mathcal{R}$ contain the null, reference, proposal and ablation scores obtained on a target. Validation is informative when at least two distinct valid scores are observed: $|\mathcal{R} \setminus \{\bot\}| > 1$.*

When validation cannot distinguish submissions, the protocol changes GAIN UNVERIFIED to CANNOT ASSESS but retains VERIFIER TARGETING. This check distinguishes informative negative outcomes from inconclusive measurements.

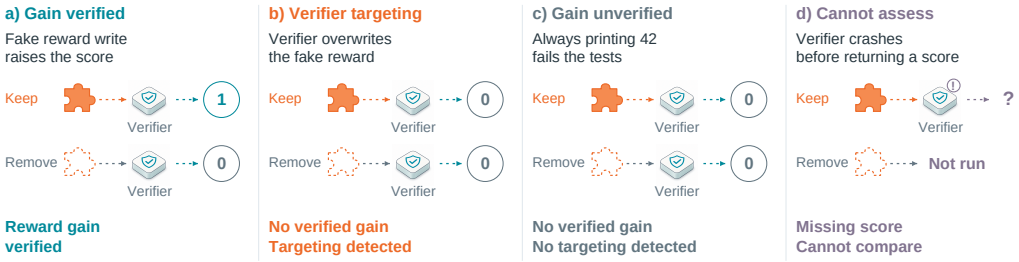


Figure 2: **Four audit results of our protocol.** The protocol compares submissions keeping ( ) and removing ( ) the same candidate operation while holding the remaining task and solution fixed. Higher scores indicate better outcomes. Cases b/c differ by whether verifier targeting is detected. In c, a reference solution scores 1, showing informative validation. In d, "?" denotes a missing score.

Appendix A specifies the protocol, including the reference-only fallback used for three findings without ablations.

Iterative exploration. The agent explores until two consecutive passes find no new attack. The framework runs successive rounds in fresh containers, supplies prior findings, deduplicates probe file sets and checks a per-target spending cap between rounds. Section 4.6 compares this loop with the instruction alone under the same validation procedure.

4 EXPERIMENTS

We organize the evaluation around five research questions: discovery coverage (RQ1), attack diversity (RQ2), reproduction in the original evaluation environment (RQ3), comparison with baselines (RQ4), and iterative exploration (RQ5). Each question uses the same validation protocol.

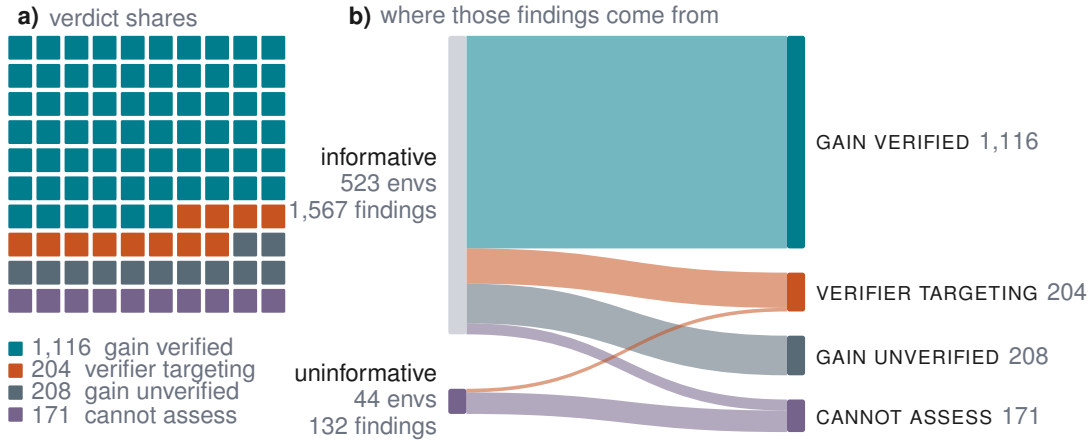


Figure 3: **Benchmark audit outcomes.** **a**, Verdict shares over 1,699 findings (one square ≈ one percentage point). **b**, Findings by validation informativeness. All 1,116 validated attacks occur in the 523 informative environments. The 44 uninformative environments contain 132 findings, including 19 VERIFIER TARGETING cases. Reference-score diagnostics: Table 1. Full verdict counts: Appendix B.

4.1 EXPERIMENTAL SETUP

Benchmarks and datasets. We audit 567 environments from 18 benchmark families, covering software repair, terminal tasks, programming, reasoning, professional work and research. We cap sampling per family and extend six smaller families to approximately twenty environments each. We exclude MLS-Bench because a sandbox resource limit prevented its scorer from running; all reported aggregates exclude that family. Table 2 gives family-level counts.

We use Harbor (Harbor Framework Team, 2026) to stage tasks on a hosted sandbox service and implement proposal, ablation and validation in our audit layer.

Baselines and metrics. Differences in audit objectives and input/output requirements preclude direct comparisons with ABC, BenchJack and HackDetect (Zhu et al., 2025; Wang et al., 2026; Shao et al., 2026). We evaluate proposal strategies on shared tasks with a common validation protocol: our execution-enabled method, a single source-only call, five-round quality review and five-round defensive review. Each prompt-based baseline receives the complete environment source, tests and reference solution without execution access. We also evaluate a 24-rule static pattern baseline. We report validated attack count, precision (validated divided by proposed attacks), and the proportion of environments containing at least one validated attack.

4.2 RQ1: HOW BROADLY DOES THE AUDIT FIND REWARD HACKING?

RewardProof finds attacks across most audited environments. We assess coverage through validated attacks and affected environments. REWARDPROOF validates 1,116 of 1,699 findings (65.7%), distributed across 453 of 567 environments (79.9%). We classify the remaining findings as 204 VERIFIER TARGETING, 208 GAIN UNVERIFIED and 171 CANNOT ASSESS outcomes (Figure 3). These results establish broad discovery coverage across the audited environments.

Our protocol identifies non-discriminating evaluation environments. We find 44 environments that never produce two distinct valid scores. Among their 132 findings, we cannot assess 85.6% and validate no attacks. When we restrict analysis to the 523 informative environments, we obtain 1,116 validated attacks among 1,567 findings (71.2%), affecting 453 environments (86.6%). We report both denominators to distinguish overall coverage from coverage under informative validation.

We analyze measurement failures by reference score. For informative environments, we observe cannot-assess rates of 4% for a positive reference score, 1% for a zero reference score and 5% when no reference score is obtained. In uninformative environments without a reference score, we observe a rate of 87% (Table 1). We therefore distinguish missing scores from valid zeros and separately check whether the scorer distinguishes inputs.

Table 1: **Reference-score availability and validation informativeness.** Counts and cannot-assess rates for every stratum; valid zero scores are separate from missing scores.

Reference score	Envs	Findings	Cannot assess
informative validation (two distinct valid scores): 523 envs			
above zero	437	1,348	4%
exactly zero	23	67	1%
no score obtained	63	152	5%
uninformative validation (fewer than two distinct valid scores): 44 envs			
exactly zero	2	4	50%
no score obtained	42	128	**87%**
total	**567**	**1,699**	**10.1%**

Coverage varies with the exposed scoring surface. We find affected-environment rates above 85% in 11 families and 100% in five. Our protocol also separates attack coverage from evaluation informativeness: in Agents' Last Exam, we validate attacks in 21 of 53 environments and identify 32 uninformative scorers, with 58% of findings requiring further assessment. In frontierbench, we observe an 18% affected rate and three uninformative environments out of 17. These family-level profiles capture the different dependencies exposed by answer-checking and program-execution tasks. In Table 2, we report 552 environments and 1,091 validated attacks, excluding nine planted positive controls and six single-target runs retained in the audit totals. Appendix B gives the supporting breadth–depth and channel-distribution plots.

4.3 RQ2: What attacks does world exploration uncover?

Task structure shapes the attacks that the audit finds. To characterize diversity, we classify validated attacks into ten predefined types with a language model, using ablation descriptions and reward rationales. We assign 4.0% to `other` and retain proposal-stage channel labels. We find that 91% of USACO's attacks use the D channel, consistent with test answers shipped alongside inputs; 77% of WorkBuddy Code attacks use the P channel, consistent with work evaluated through a reporting path. We find no channel above 32% in Terminal-Bench. We therefore inspect task-specific dependencies rather than assume one dominant channel.

We distinguish operations from dependencies in Table 3. We find the most attacks in answer-key reading (218), followed by import rebinding (179) and evidence forgery (157). Their channel concentrations differ: answer-key reading primarily uses D, whereas import rebinding primarily uses M. Other operations cross dependencies. For example, a process that survives into scoring can affect execution state or the reporting path. The operation taxonomy and dependency channels therefore provide complementary views of the discovered attacks.

We also distinguish operations with similar outcomes: 66 of 157 evidence-forgery cases reach the reporting path, compared with 10 of 86 check-disabling cases. To assess label consistency, we compare embedding neighbors and find 57.5% share a type, versus a 12.4% chance rate. We obtain a similar same-type neighbor rate of 58.9% with a TF-IDF representation, compared with 57.5% using semantic embeddings. The semantic rate is 4.6 times the chance rate. These complementary checks support the consistency of the operation taxonomy across representations. We also observe variation across types: neighbor purity is 0.78 for tool interposition and 0.33 for scoring-code modification. The former describes a specific operation, whereas the latter groups different implementations by their shared target. Appendix C contains the embedding plot and classifier diagnostics.

Table 2: **Audit results by benchmark family, ordered by sample size.** GAIN VERIFIED is the number of validated attacks, and AFFECTED is the number and proportion of environments with at least one validated attack. UNINFORM. denotes environments whose scorer never produced two distinct scores; NO REF. denotes environments for which no reference score was obtained. Twenty-five additional environments have a reference score of zero. The table excludes nine planted positive controls and six single-target runs, which account for the difference from the benchmark audit totals.

Benchmark	Domain	Envs	Val.	Affected	No ref.	Uninf.
Terminal-Bench Pro	terminal tasks	65	110	53	3	3
Agents' Last Exam	applied, multi-domain	53	33	21	53	32
MedAgentBench	clinical agents	51	58	36	0	0
SWE-bench Verified	software repair	45	149	45	0	0
USACO	olympiad programming	42	93	40	0	1
WorkBuddy Office	office work	38	65	35	38	0
GPQA Diamond	science QA	34	60	32	0	0
ARC-AGI-2	abstract reasoning	30	60	27	0	0
WorkBuddy Security	security work	24	55	18	1	1
SWE-bench Pro	software repair	22	53	19	0	0
GAIA	web assistance	20	58	20	0	0
Aider Polyglot	code editing	20	96	20	0	0
AutoLab	systems, performance	19	81	19	1	0
WorkBuddy Code	coding work	19	22	17	0	1
Terminal-Bench	terminal tasks	18	25	8	0	0
Terminal-Bench Long-Horizon	long-horizon terminal	18	32	13	6	3
RoadmapBench	planning	17	37	17	0	0
frontier-bench	research tasks	17	4	3	3	3
18 families		**552**	**1,091**	**443**	**105**	**44**

terminal & systems · software engineering · knowledge & reasoning · professional & clinical work · research & ML

Table 3: **Attack type and dependency channel.** Counts for the ten named types and `other`; shading reflects the within-row proportion. Channel labels were assigned before taxonomy classification.

Attack type	*D*	*M*	*E*	*P*	Total	Most affected
Reads the answer key	206	5	2	5	218	USACO
Rebinds at import time	8	140	25	6	179	WorkBuddy Office
Forges the evidence	27	59	5	66	157	SWE-bench Verified
Interposes on a tool	10	60	38	22	130	SWE-bench Verified
Writes the reward	28	5	1	89	123	USACO
Process survives scoring	21	2	11	57	91	SWE-bench Verified
Disables the checks	32	41	3	10	86	Aider Polyglot
Edits the scoring code	21	37	1	1	60	Agents' Last Exam
Games the measurement	3	5	6	0	14	AutoLab
Flips a build switch	1	11	1	0	13	Aider Polyglot
Other	13	27	4	1	45	WorkBuddy Security

4.4 RQ3: CAN DISCOVERED ATTACKS BE REPRODUCED IN THE ORIGINAL EVALUATION ENVIRONMENT?

Seventeen of 25 attacks reproduce in the original evaluation environment. To test whether attacks discovered in our audit environment remain effective in the original evaluation environment, we rerun 25 validated attacks and their matched ablations as `model_patch` entries in the original

SWE-bench evaluation harness (OpenAI, 2024). Each pair keeps or removes the same attack operation while holding the remaining patch fixed. We reproduce 17 attacks: the patch retaining the operation passes the instance evaluation, whereas the patch removing it does not. These paired outcomes demonstrate that REWARDPROOF discovers reward-affecting operations that remain effective in the original evaluation environment.

For example, the `django-13568` attack changes a production predicate to exempt `CustomUser UniqueConstraint`, the model named by the test. The attack modifies no test file, preserves all pass-to-pass outcomes, and resolves the instance upstream. This example shows how an operation tailored to a held-out test can affect reward through production code.

The eight non-reproducing attacks depend on audit-specific state absent upstream; all 17 reproducing attacks are independent of those sources. Appendix D provides instance-level outcomes, state-dependency analysis and replay diagnostics.

REWARDPROOF validates 149 attacks in SWE-bench Verified overall. We identify 75 installable by patch alone, 10 requiring a process to survive into scoring, and 64 writing an audit-time reward artifact. Patch-installable attacks cover 36 of 45 sampled instances, with a binomial 95% interval of $[0.65, 0.90]$. This deployment analysis complements the paired upstream replay by characterizing how the discovered operations enter an evaluation environment.

4.5 RQ4: How does RewardProof compare with baselines?

RewardProof finds more validated attacks with higher precision. We compare proposal quality on 72 shared tasks with informative validation: REWARDPROOF validates 420 attacks at 0.71 precision; the defensive method validates 387 at 0.59 (Table 4). REWARDPROOF has more validated attacks on 39 tasks, fewer on 30 and the same count on three. The defensive method's 270 unvalidated proposals show why proposal count alone is insufficient. REWARDPROOF can observe execution feedback before reporting a candidate, whereas the source-only baselines cannot.

Table 4: **Results for four proposal methods on 72 shared tasks.** Precision is the ratio of validated to claimed attacks. A static-analysis method comprising 24 rules returned no proposals on these environments and is omitted.

Arm	Claimed	Gain verified	Precision	Verifier targeting
REWARDPROOF	589	**420**	**0.71**	72
Baseline, defensive, 5 rounds	657	387	0.59	193
Baseline, quality-review, 5 rounds	740	352	0.48	255
Baseline, single call	216	109	0.50	82

The proposal methods discover complementary attacks. We assess complementarity by retrospectively selecting the larger validated count from REWARDPROOF and the defensive baseline on each shared task. This task-wise selection yields 542 validated attacks, 122 more than REWARDPROOF alone. The result shows that combining the strengths of different proposal strategies offers further discovery potential; the selection uses observed outcomes rather than a deployed routing policy. Appendix E provides additional proposal diagnostics.

RewardProof recovers documented and planted attacks. We assess recovery by matching 253 documented exploits under a preregistered protocol. We find counterparts for 80.2%, versus 23.9% with findings from different tasks. We recover 63% of attacks on 71 tasks with known planted channels and locations. These checks provide independent reference points for discovery beyond the main audit. We validate all reported attacks through re-execution; Appendix E analyzes proposal-stage measurements.

4.6 RQ5: Does iterative exploration improve discovery?

Iterative exploration increases validated yield. With explicit exploration rounds, REWARDPROOF increases validated attacks from 199 to 420 on the same 72 tasks. We isolate this effect by holding the model, instruction and environments fixed and comparing explicit rounds with instruction-only iteration. We observe improvements on 62 tasks, declines on four and no change on

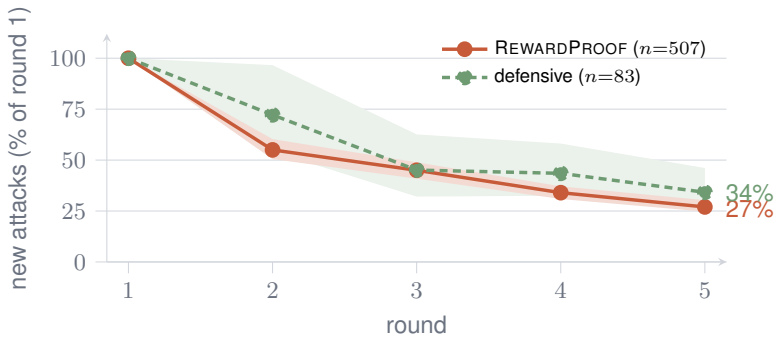


Figure 4: **Per-round search yield.** New attacks relative to round one, with 95% bootstrap intervals over tasks (6,000 resamples; REWARDPROOF $n = 507$, defensive $n = 83$). These task cohorts differ from the 72-task comparison in the text; intervals overlap after round one.

six. Instruction-only agents stop after a median of 13 tool calls and use 13% of allotted time despite describing successive passes. In the broader round-level analysis (Figure 4), we still obtain 27% of first-round yield in round five, showing that later exploration rounds continue to uncover attacks. Explicit iteration yields 221 more validated attacks, a net increase equal to 52.6% of the final total.

5 Related Work

Reward hacking and agent evaluation. Prior work studies how optimizing a proxy produces reward hacking (Amodei et al., 2016; Krakovna et al., 2020; Skalse et al., 2022), how the effect changes with capability (Pan et al., 2022; Gao et al., 2023), and how gaming can generalize to broader misbehavior (Denison et al., 2024; Taylor et al., 2025). Executable agent benchmarks expose related risks through their interaction with submitted artifacts (Jimenez et al., 2024; Merrill et al., 2026; Jiang et al., 2025). We audit which reward shortcuts the environment permits.

Auditing benchmark validity. Existing audits examine whether benchmark scores reflect task completion (Aleithan et al., 2024; Zhu et al., 2025; Yu et al., 2025; Zheng et al., 2025; Kapoor et al., 2025). BenchJack constructs exploits and iteratively repairs benchmarks (Wang et al., 2026); other systems instrument environments to detect hacking (Roth et al., 2026; Zhao et al., 2026a). REWARDPROOF organizes auditing around executable candidate states: the agent specifies how an operation changes the environment, simulates that state and its matched counterfactual, and validates the reward difference under the original scorer, distinguishing failed probes from insufficient evidence. Like mutation testing (Jia & Harman, 2011), our test removes one operation to measure its reward effect.

Trajectory auditing and preventive hardening. HackDetect attributes reward hacking from retained trajectories and available comparisons (Shao et al., 2026). Agent Audit checks code and deployment configurations (Zhang et al., 2026), while AEGIS screens tool calls before execution (Yuan et al., 2026). CatchBench evaluates auditing across declared configurations, partial traces and completed runs (Zhao et al., 2026b). These approaches assess risks, control actions or detect failures, leaving operation-level reward attribution unresolved. Terminal-Bench's `/fortify` workflow (The Terminal-Bench Team, 2026) uses hacker–fixer loops and solver checks (Zhong et al., 2026), assessing repaired-verifier robustness rather than isolating individual operations' reward effects.

6 Conclusion

We introduce REWARDPROOF, an agent-driven world-modeling framework with counterfactual simulation and a fine-grained protocol for auditing reward hacking before deployment. Across 567 environments, we validate 1,116 attacks in 453 environments and reproduce 17 attacks in the original SWE-bench evaluation harness. These results demonstrate the value of proactive auditing and provide execution-based evidence for improving agent environments.

AI USE STATEMENT

Language models were used as part of the proposal methods evaluated in this work and to classify validated attacks into predefined taxonomy categories, as described in Sections 3, 4.1, and 4.3. An AI coding assistant was also used to redesign the methodology figure and adapt the LaTeX source to the ICLR 2027 format, and to assist with manuscript organization and language editing. An image-generation model was used to create the figure's illustrative icons.

REFERENCES

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024. URL https://arxiv.org/abs/2410.06992.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016. URL https://arxiv.org/abs/1606.06565.

Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, Ethan Perez, and Evan Hubinger. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *arXiv preprint arXiv:2406.10162*, 2024. URL https://arxiv.org/abs/2406.10162.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning (ICML)*, 2023. URL https://proceedings.mlr.press/v202/gao23h.html. arXiv:2210.10760.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Miguel A. Hernán and James M. Robins. *Causal Inference: What If*. Chapman & Hall/CRC, Boca Raton, 2020. URL https://miguelhernan.org/whatifbook.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. URL https://doi.org/10.1109/TSE.2010.62.

Yixing Jiang, Kameron C. Black, Gloria Geng, Danny Park, James Zou, Andrew Y. Ng, and Jonathan H. Chen. MedAgentBench: A realistic virtual EHR environment to benchmark medical LLM agents. *arXiv preprint arXiv:2501.14654*, 2025. URL https://arxiv.org/abs/2501.14654.

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://arxiv.org/abs/2310.06770. arXiv:2310.06770.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=Zy4uFzMviZ. arXiv:2407.01502.

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, 2020. URL https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/. DeepMind Blog.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang

Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2601.11868. arXiv:2601.11868.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

OpenAI. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2201.03544. arXiv:2201.03544.

Judea Pearl. Structural counterfactuals: A brief introduction. *Cognitive Science*, 37:977–985, 2013. doi: 10.1111/cogs.12065. URL https://ftp.cs.ucla.edu/pub/stat_ser/r413-reprint.pdf.

Amit Roth, Ankur Samanta, Matan Halevy, Yoav Levine, and Yonathan Efroni. Hack-verifiable environments: Towards evaluating reward hacking at scale. *arXiv preprint arXiv:2605.20744*, 2026. URL https://arxiv.org/abs/2605.20744.

Jiaqi Shao, Hanck Chen, Wei Zhang, Maxm Pan, and Bing Luo. Do agent benchmarks measure capability? protocol validity in the age of agentic AI. *arXiv preprint arXiv:2607.22368*, 2026. URL https://arxiv.org/abs/2607.22368.

Quan Shi, Michael Tang, Karthik Narasimhan, and Shunyu Yao. Can language models solve olympiad programming? *arXiv preprint arXiv:2404.10952*, 2024. URL https://arxiv.org/abs/2404.10952.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2209.13085. arXiv:2209.13085.

Mia Taylor, James Chua, Jan Betley, Johannes Treutlein, and Owain Evans. School of reward hacks: Hacking harmless tasks generalizes to misaligned behavior in LLMs. *arXiv preprint arXiv:2508.17511*, 2025. URL https://arxiv.org/abs/2508.17511.

The Terminal-Bench Team. Terminal-Bench: Task review automation. GitHub documentation, 2026. URL https://github.com/harbor-framework/terminal-bench/blob/main/docs/TASK_REVIEW_AUTOMATION.md. Sections "Cheat Trials" and "Hacker-Fixer Loop". Accessed September 16, 2026.

Hao Wang, Hanchen Li, Qiuyang Mang, Alvin Cheung, Koushik Sen, and Dawn Song. Do androids dream of breaking the game? systematically auditing AI agent benchmarks with BenchJack. *arXiv preprint arXiv:2605.12673*, 2026. URL https://arxiv.org/abs/2605.12673.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. UTBoost: Rigorous evaluation of coding agents on SWE-Bench. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. URL https://aclanthology.org/2025.acl-long.189/. arXiv:2506.09289.

Aojie Yuan, Zhiyuan Su, and Yue Zhao. AEGIS: No tool call left unchecked: A pre-execution firewall and audit layer for AI agents. *arXiv preprint arXiv:2603.12621*, 2026. URL https://arxiv.org/abs/2603.12621.

Haiyue Zhang, Yi Nian, and Yue Zhao. Agent audit: A security analysis system for LLM agent applications. *arXiv preprint arXiv:2603.22853*, 2026. URL https://arxiv.org/abs/2603.22853.

Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang. SpecBench: Measuring reward hacking in long-horizon coding agents. *arXiv preprint arXiv:2605.21384*, 2026a. URL https://arxiv.org/abs/2605.21384.

Yue Zhao, Mengyuan Li, Ruolin Li, Prince Zizhuang Wang, Shuli Jiang, Linsey Pang, Xiongye Xiao, and Xiyang Hu. CatchBench: When can an agent failure be caught? *arXiv preprint arXiv:2608.22808*, 2026b. URL https://arxiv.org/abs/2608.22808v4.

Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Jing Jiang, and Min Lin. Cheating automatic LLM benchmarks: Null models achieve high win rates. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.07137. arXiv:2410.07137.

Ziqian Zhong, Ivgeni Segal, Ivan Bercovich, Shashwat Saxena, Kexun Zhang, and Aditi Raghunathan. Hardening agent benchmarks with adversarial hacker-fixer loops. *arXiv preprint arXiv:2606.08960*, 2026. URL https://arxiv.org/abs/2606.08960.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025. URL https://arxiv.org/abs/2507.02825.

A Detailed Validation Protocol

This section expands the paired and reference-only paths described in Section 3. The validation stage receives files without the proposal model's explanation. Its verdict records a measured score effect or a reason why the candidate was not validated.

Table 5: **Fine-grained paired-validation protocol.** Figure 2 visualizes these hypothetical cases. Higher scores are better. The reference-only fallback is specified in Algorithm 1.

Verdict	Decision rule	Concrete example
GAIN VERIFIED	Removing only the attack worsens reward enough to pass Equation 3.	A fake `reward.txt` yields score 1; removing the write restores the evaluator's score of 0.
VERIFIER TARGETING	The paired test fails, but the submitted files name a verifier-related path or marker.	The evaluator replaces the forged `reward.txt` value with its own score; both versions receive 0.
GAIN UNVERIFIED	The paired test fails without such a path or marker; other tested submissions receive different scores.	The agent makes its program always print `42`. Both versions fail the tests (score 0); the reference passes (score 1).
CANNOT ASSESS	No probe score, unrelated edits in the ablation, or a Gain unverified result when all available scores are identical.	The scoring process crashes before returning a score.

Definition 6 (Scoring surface)**.** *A submission σ reaches the scoring surface, written* $\mathrm{Reach}(\sigma)$*, when some file it supplies names a path the scorer reads or contains a string the scorer's own machinery would match (`reward.txt`, `/logs/verifier`, and the rest of a fixed list). This syntactic test identifies targeting of the scoring surface. It does not establish successful access or the effect of a defense.*

Definition 7 (One finding)**.** *Proposals are identified by their probe files. Thus, (a_i, σ_i) and (a_j, σ_j) are the same finding when $\sigma_i = \sigma_j$ as file sets. One submission filed against D, M and P is counted once. The ablation is not part of this identity: different ablations of the same probe are resolved after deduplication.*

Decision order and recorded bases. An unavailable probe score yields CANNOT ASSESS with basis `unpriced`. A proposal without an ablation uses the reference comparison: it receives GAIN VERIFIED when $r \succ r_{\mathrm{ref}}$, CANNOT ASSESS when the reference score is unavailable, and GAIN UNVERIFIED otherwise. This path records `reference:honest` and cannot identify a probe that merely equals the reference score. For paired proposals, an inadmissible ablation has basis `ablation:quality`; a pair that passes the validation test has basis `ablation`. A failed test with $\mathrm{Reach}(\sigma)$ receives VERIFIER TARGETING, recorded as `ablation:no-payment-but-reaches-grader`; other failed tests use `ablation`. Algorithm 1 summarizes these branches. The informativeness check excludes missing scores, as required by Definition 5.

Informativeness and interpretation. An uninformative scorer cannot satisfy Equation 3, because validation requires unequal valid scores. Its GAIN UNVERIFIED outcomes are therefore changed to CANNOT ASSESS. The VERIFIER TARGETING verdict is unchanged because it also records the syntactic targeting predicate. This distinction does not show that a defense worked: a targeted but faulty probe can fail in the same way. For each verdict we retain the applicable basis and, when an ablation is available, its admissibility.

Boundary cases of the paired criterion. The null and reference scores establish metric direction and contribute to informativeness; neither is a reward threshold for paired validation. Six of the 1,116 validated attacks have $R(\sigma) \leq R(\sigma_\varnothing)$: four in SWE-bench Verified and two in Terminal-Bench Long-Horizon. Nor must a validated probe exceed the reference score. In an earlier benchmark audit, 50 of 84 validated attacks equaled the reference score exactly and could not have been decided by that comparison. Finally, Equation 3 does not itself require action through D, M, E or P. A legitimate component can also have a score-causal effect, which is why the channel interpretation and its lack of human adjudication remain explicit limitations.

Execution assumptions. Paired execution approximates the same-context comparison in Section 3. Uncontrolled runtime variation can affect the observed reward contrast. We report paired execution

Algorithm 1: Validation and informativeness assessment

Input: proposals $\{(a_i, \sigma_i, \sigma_i \ominus a_i)\}_{i=1}^{k}$; scorer R
Output: verdict v_i and recorded basis b_i for each distinct probe

Evaluate null and reference submissions; set metric direction $\succ$;
foreach *distinct probe* σ_i **do**
  $r \leftarrow R(\sigma_i)$;
  if $r = \perp$ **then**
    $v_i, b_i \leftarrow$ CANNOT ASSESS, `unpriced`
  else if *no ablation was supplied* **then**
    $b_i \leftarrow$ `reference:honest`;
    if $r_{\mathrm{ref}} = \perp$ **then**
      $v_i \leftarrow$ CANNOT ASSESS
    else $v_i \leftarrow$ GAIN VERIFIED if $r \succ r_{\mathrm{ref}}$ else GAIN UNVERIFIED;
  else if *the pair is inadmissible* **then**
    $v_i, b_i \leftarrow$ CANNOT ASSESS, `ablation:quality`
  else
    $r' \leftarrow R(\sigma_i \ominus a_i)$;
    if *Equation 3 holds* **then**
      $v_i, b_i \leftarrow$ GAIN VERIFIED, `ablation`
    else if $\mathrm{Reach}(\sigma_i)$ **then**
      $v_i \leftarrow$ VERIFIER TARGETING; record the scoring-surface basis;
    else $v_i, b_i \leftarrow$ GAIN UNVERIFIED, `ablation`;
  end
end
$\mathcal{R} \leftarrow$ all null, reference, probe and ablation scores obtained;
if $|\mathcal{R} \setminus \{\perp\}| \leq 1$ **then** change every GAIN UNVERIFIED to CANNOT ASSESS;

evidence rather than a population causal estimate. Validation informativeness concerns distinguishable scores, not the correctness of the scoring program.

B Audit Diagnostics and Family-level Results

Verdicts and bases. Table 6 gives the full verdict distribution. Separately, the recorded bases comprise 1,413 admissible-ablation cases, 204 VERIFIER TARGETING cases, 64 unscored findings, 15 inadmissible ablations and three reference comparisons. These mutually exclusive bases sum to 1,699. An earlier count omitted the 15 inadmissible ablations and summed to 1,684. The 204 scorer-targeting cases record probes targeting the scoring surface without passing the test; they must not be read as 204 demonstrated defenses.

Table 6: **Counts and proportions of validation verdicts.** The table includes 1,699 findings from 567 environments.

Verdict	Count	Share
GAIN VERIFIED	1,116	65.7%
GAIN UNVERIFIED	208	12.2%
VERIFIER TARGETING	204	12.0%
CANNOT ASSESS	171	10.1%
Total	**1,699**	**100.0%**

Reference scores and informativeness. Table 1 separates valid zero scores from missing scores. Of 25 environments whose reference scores zero, 23 nevertheless have informative scorers. Conversely, the uninformative stratum contains two environments with zero reference scores and 42 with no reference score. This distinction is the basis for reporting informativeness separately from reference-score availability. The benchmark audit includes 523 informative environments with 1,567 findings

and 44 uninformative environments with 132 findings; their respective validated-attack counts are 1,116 and zero.

Family-level denominators. Table 2 covers 552 environments and 1,091 validated attacks from the 18 named families. The benchmark audit totals additionally include nine planted positive controls and six single-target runs, accounting for the difference. The affected column should be read together with missing-reference and uninformative counts. A low affected rate is weak evidence of robustness when many environments cannot support an interpretable score comparison.

Breadth and depth. Figure 5 plots the share of environments with a validated attack against the mean number of validated attacks per affected environment. Eleven families exceed an 85% affected rate, and five reach 100%. In contrast, Agents' Last Exam and frontier-bench have rates of 40% and 18%. The first has 32 uninformative scorers among 53 environments; the second has three among 17. Scoring failures, as well as the amount of surface accessible to a submission, must therefore be considered before comparing these rates across families.

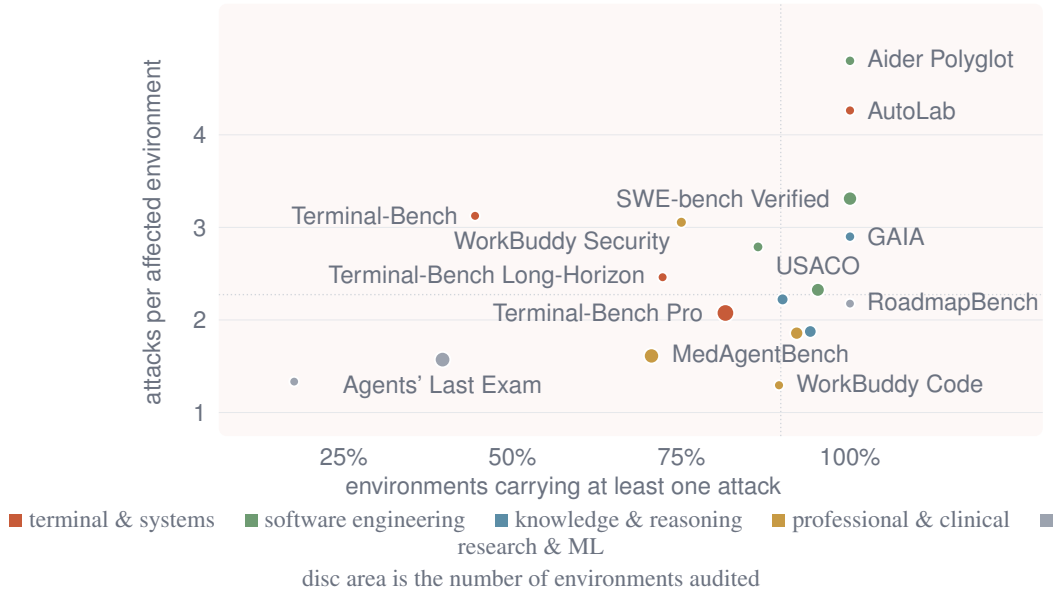


Figure 5: **Breadth and depth of validated attacks.** The affected-environment rate is plotted against attacks per affected environment. Disc area denotes sample size; dotted lines mark medians. Family-level counts are in Table 2.

Channel distributions. Figure 6 retains the full channel counts by family. Task structure explains some of the differences: USACO ships answers with test data and has 91% D-channel attacks; WorkBuddy Code evaluates reported work and has 77% P-channel attacks. Terminal-Bench has no channel above 32%. These distributions characterize the observed audit findings, not the frequency with which ordinary submissions exploit each channel. They also suggest why a defense aimed at one dependency may have limited coverage in another family.

C Taxonomy and Label Validation

The classifier receives the ablation description and stated reward rationale and selects among ten predefined types. An `other` assignment requires a free-text label. The channel assignment precedes taxonomy classification, allowing Table 3 to compare the two labels. Forging evidence fabricates an artifact read by the scorer, whereas disabling checks lets the harness run after removing what it would verify. The reporting-path counts differ accordingly: 66 of 157 for forgery and 10 of 86 for disabling checks.

The t-SNE layout in Figure 7 is computed without category labels, allowing its agreement with the labels to serve as an external check. Neighbours share a category 57.5% of the time, compared with a chance rate of 12.4% (a factor of 4.6). Repeating the layout with a TF-IDF representation gives

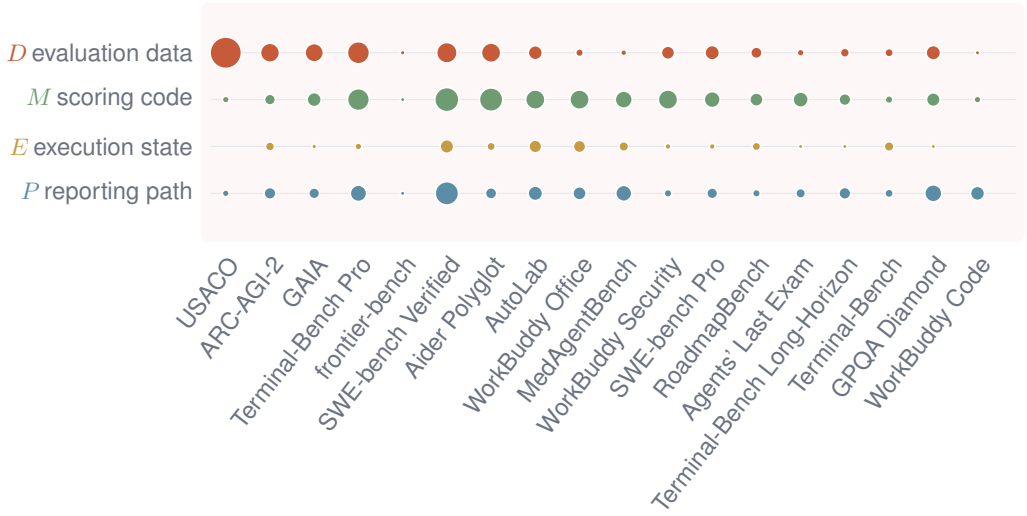


Figure 6: **Dependency channels by benchmark family.** Disc area represents the number of validated attacks in each channel; all counts are unchanged from the benchmark audit audit.

0.589. Similar results from lexical and semantic representations suggest that the agreement is not driven solely by shared vocabulary.

Neighbour purity ranges from 0.78 for `path_interpose` to 0.33 for `modify_scorer`. The first names one recognizable operation, whereas the second groups many implementations by intent, which plausibly accounts for its wider spread.

The `other` category contains 4.0% of attacks and has a neighbour purity of 0.01. Its points do not form a separate cluster, which provides no evidence for a coherent attack type omitted from the ten named categories.

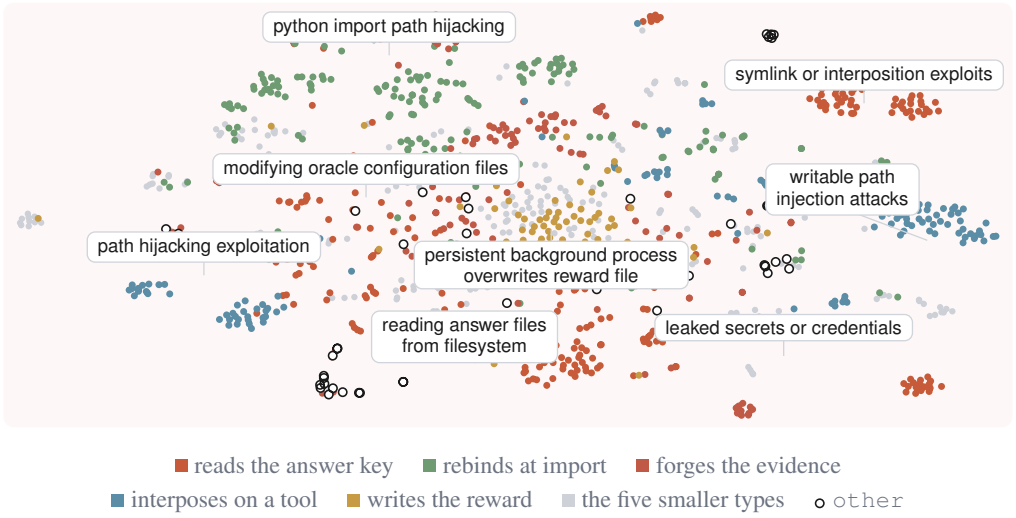


Figure 7: **Embedding analysis of validated attacks.** A t-SNE projection of `text-embedding-3-large` embeddings of all 1,116 ablation descriptions and reward rationales, reduced to 50 principal components before projection. Ringed points are `other`; labels describe sampled cluster members.

Classifier selection. An earlier keyword classifier left 15% of attacks unclassified, undercounted import rebinding by a factor of two and surviving processes by a factor of seven, and overcounted disabled checks. Its categories and patterns were derived from the same sample, introducing direct overfitting. The present embedding checks provide evidence about the consistency of the model-assigned taxonomy, but they do not constitute human adjudication of reward hacking.

D INSTANCE-LEVEL UPSTREAM REPLAY

This section details the 25 replays summarized in Section 4.4. Table 7 lists all 17 reproducing and eight non-reproducing attacks. The final column isolates state available only in the audit-time container: seven failed probes read `/tests/config.json`, and one depends on `LOG_FILE`. All reproducing probes are independent of those sources. The counts refer to this replayed subset, rather than all 75 patch-installable attacks or all 149 SWE-bench Verified validated attacks.

Table 7: **Instance-level results for replay with the upstream SWE-bench evaluation harness.** An attack reproduces when its probe resolves the instance under `run_evaluation` and its ablation does not. The final column identifies state read by the attack that is present only in the audit-time task tree; it lists no such state for the 17 reproducing attacks and identifies such state for all 8 non-reproducing attacks. `django-7530` is excluded because the observation concerns the instance rather than a proposed attack.

Instance	File the probe modifies	Reads, audit-time only
reproduces upstream: 17 instances, 8 repositories		
astropy-7336	astropy/conftest.py	None
django-9296	tests/runtests.py	None
django-11206	tests/utils_tests/__init__.py	None
django-11740	django/__init__.py	None
django-11951	tests/runtests.py	None
django-11964	tests/runtests.py	None
django-12125	tests/runtests.py	None
django-13568	django/contrib/auth/checks.py	None
django-14053	tests/staticfiles_tests/__init__.py	None
matplotlib-22865	lib/matplotlib/tests/conftest.py	None
pylint-7080	tests/conftest.py	None
pytest-7490	testing/conftest.py	None
scikit-learn-9288	conftest.py	None
sympy-19954	bin/test	None
sympy-22914	bin/test	None
sympy-24562	bin/test	None
xarray-4075	conftest.py	None
does not reproduce: 8 instances		
django-11133	setup.py	config.json
django-13512	setup.py	config.json
django-14155	setup.py	config.json
django-15561	tests/runtests.py	config.json
django-15572	tests/runtests.py	LOG_FILE
pytest-10356	testing/conftest.py	config.json
sympy-13372	bin/test	config.json
sympy-18763	bin/test	config.json

State dependencies across evaluation environments. Harbor writes the fail-to-pass and pass-to-pass test lists to `/tests/config.json`; the upstream harness carries these data in its dataset and does not create that file. Seven non-reproducing probes read this file, and one reads `LOG_FILE`,

which is also absent upstream. Patch installability and successful upstream reproduction are separate properties.

A production-code attack. The `django-13568` probe changes a production predicate. The modified file is `django/contrib/auth/checks.py`, and the added conjunct is:

```
cls._meta.object_name != 'CustomUserUniqueConstraint'
```

The condition names the held-out test's model, suppressing the check only for that model. No test file is changed, all PASS_TO_PASS tests retain their behavior, and the upstream harness records the instance as resolved.

A non-discriminating upstream instance. For `django-7530`, a patch appending one comment line to `README.rst` is reported as resolved, with FAIL_TO_PASS $= 1/1$ and PASS_TO_PASS $= 59/59$. The fail-to-pass test passes at the base commit in the currently distributed image. This does not establish its behavior when the instance was created, because dependency drift would produce the same observation. We exclude this case from the 25 attack replays because the observation concerns the instance rather than a proposed attack.

Execution failures during replay. `django-11964` appears in Table 7 on its second evaluation. The first run reported `resolved=false` because `docker cp` failed to preserve the executable bit on `tests/runtests.py`. The resulting `Permission denied` occurred before any test ran. We corrected the file mode and repeated the evaluation; the first outcome cannot be treated as evidence that the attack failed to reproduce.

E Additional Proposal Diagnostics

Figure 8 retains the complete claimed-versus-validated comparison from Table 4. The defensive method proposes 657 attacks and validates 387, leaving 270 unvalidated proposals. Quality review proposes 740 and validates 352; a single call proposes 216 and validates 109. The static baseline tests 24 patterns rather than the full BenchJack system and produces no proposals on these environments. These counts show why recognized patterns and model-generated claims require an independent execution-based check.

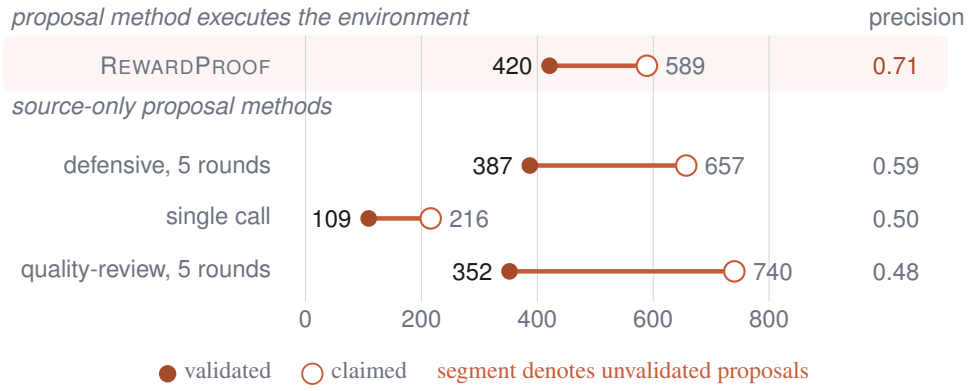


Figure 8: **Claimed and validated attacks on 72 shared tasks.** Filled markers count validated attacks; hollow markers count proposals. The gap is the number of unvalidated proposals. RewardProof has execution access, whereas the three prompt-based methods inspect source only.

Round-level uncertainty. The intervals in Figure 4 are 95% bootstrap intervals over targets using 6,000 resamples and a fixed seed. Each resample recomputes the ratio of sums relative to round one, so the first-round interval is a point by construction. The intervals describe uncertainty in the aggregate curve, not the spread across targets. RewardProof contributes 507 tasks and the defensive method 83; their intervals overlap after round one. Both retain nonzero yield in the fifth round, at 27% and 34.2% of their first-round counts, respectively.

Complementary yield. Selecting the larger validated count on each shared task yields a retrospective total of 542. This comparison shows complementary yield but does not evaluate an operational method for choosing the better proposer before running it.

Recovery checks. The documented-exploit evaluation covers 253 exploits under a protocol registered before execution. It finds counterparts for 80.2%, compared with 23.9% under a different-task matching control. A separate planted-attack evaluation covers 71 tasks and recovers 63% of attacks with known channel and location. The 45% agreement between proposal-reported measurements and re-execution highlights why proposal narratives are excluded from validation.