WHO VERIFIES THE AGENTS? AUDITING MECHANICAL EVALUATION IN AN INFRASTRUCTURE-REPAIR ENVIRONMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Executable agent environments can verify terminal states, but terminal verification alone does not ensure that an episode validly tests diagnosis and repair. We audit CrashDiag, an infrastructure-repair environment with 52 multi-fault workflows and mechanically checked state transitions. The audit identifies a scenario-generation defect: a history action intended as an unsuccessful remediation partially repairs an active hidden sub-fault before the policy receives its observation. Reconstructing 192 retained held-out composite episodes from their seeds shows a hard boundary: the relevant sub-fault is resolved in all 117 noisy or shifted-noisy episodes and in none of the 75 redacted episodes. Thus, 34 of 228 retained Group Relative Policy Optimization (GRPO) exact successes and 8 of 16 base-policy exact successes are affected by pre-inference state leakage. We invalidate the prior policy comparison rather than post-hoc adjusting it. We formalize a pre-inference task-integrity invariant and give an audit protocol, regression-test requirements, and reporting requirements for mechanically verified agent environments.

1 INTRODUCTION

Executable environments make it possible to assess agents by state transitions rather than surface-form answers. This matters for software engineering and operational repair, where a plausible action sequence is insufficient if the system remains unhealthy. Agent benchmarks have advanced realistic, end-to-end evaluation (Liu et al., 2024; Zhou et al., 2024; Jimenez et al., 2024). Yet a terminal verifier establishes only that the final state satisfies a predicate. It does not establish that the policy, rather than scenario construction, caused the relevant repair.

We audit CrashDiag, a controlled environment for infrastructure diagnosis and remediation. CrashDiag has explicit fault workflows, a bounded action interface, and a mechanical verifier. Its retained results initially appeared to support a reinforcement-learning comparison. We found that the episode generator can append a supposedly unsuccessful historical action which repairs part of an active workflow before inference. The finding is not merely noisy: exact success is absent in every reconstructed redacted event and appears only where the leaked partial state is present.

Our contributions are:

- We distinguish terminal-state verification from *pre-inference task integrity*, and formulate an invariant requiring all active faults to remain unresolved when the policy first acts.
- We reconstruct retained evaluation episodes and identify a generator defect that pre-resolves an OOM-related sub-fault in 117 of 192 affected composite episodes.
- We invalidate, rather than cosmetically correct, the retained base-versus-GRPO comparison; clean data and a fresh matched run are required.
- We propose generator-level tests and reporting requirements that make state leakage detectable before agent results are reported.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

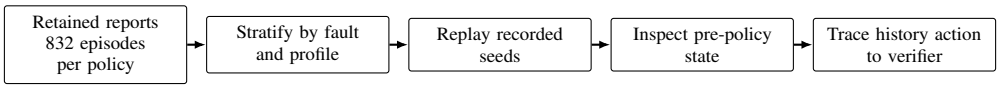

Figure 1: Discovery and confirmation path. Aggregate disparity generated a hypothesis; deterministic seed replay and source-level state tracing established the mechanism.

2 Environment and verification

CrashDiag contains 52 deterministic fault workflows. A workflow may contain multiple sub-faults; the policy receives an observation and can issue up to eight actions from an interface of 27 operations. The environment supports controlled mock, HTTP, and Docker-backed execution modes. It does not claim to reproduce production infrastructure.

The final verifier reports the number C of resolved active sub-faults among N required sub-faults. The retained reward and exact-success indicator are

$$r = \frac{C}{N}, \qquad s = \mathbb{1}[C = N].$$

These are suitable terminal quantities, but neither constrains the state at which the policy starts.

3 Pre-inference task integrity

Let F_{active} denote an episode's active sub-faults and let x_{pre} be the sandbox state immediately before policy inference. We require

$$\forall f \in F_{\text{active}}, \qquad \text{resolved}(f, x_{\text{pre}}) = 0.$$

For tasks intended to begin unhealthy, the generator should also assert that the workflow has not already reached its terminal state. This check must run after every scenario mutation, including history injection, profile perturbations, retries, and observation rendering.

A safe construction sequence is:

$$\begin{array}{ccccc} \text{Construct baseline} & \rightarrow & \text{Inject faults} & \rightarrow & \text{Add history} \\ \text{Policy access} & \leftarrow & \text{Render observation} & \leftarrow & \text{Assert integrity} \end{array}$$

State-neutrality must be checked programmatically, not inferred from an action's name. A historical action can be harmless for one literal fault but repair a sub-fault embedded in another workflow.

4 Audit methodology

We audited two retained 832-episode reports, one for a base policy and one for a GRPO-trained policy. The held-out set spans 52 workflow names with 16 seeded episodes each. Both reports refer to the same held-out JSONL (SHA-256 `2f5e8f57...e8fd169`), but the base evaluator also used a format demonstration and a lower output-token limit. We therefore use the reports to trace the integrity defect, not to estimate a causal training effect.

4.1 How the defect was found

The audit began with the full 52×16 per-fault outcome matrix and the complete ordered action lists for both policies. Large fault-level disparities motivated stratification by the recorded `scenario_profile`. Exact successes for composite workflows containing OOM kill appeared only in noisy profiles, which add a historical remediation after fault injection. We then reconstructed each episode from its retained seed, inspected the raw state immediately before inference, and compared redacted with noisy profiles. Finally, we traced the last history event backward through the scenario generator and forward through the OOM verifier. Figure 1 summarizes this chain; every arrow corresponds to a retained artifact or executable source transition.

Table 1: Reconstructed held-out episodes for the 12 affected composite workflows. Pre-policy state is measured immediately before observation and inference.

Profile group	Episodes	OOM resolved	Base exact	GRPO exact
Redacted	75	0/75	0/75	0/75
Noisy or shifted-noisy	117	117/117	8/117	34/117

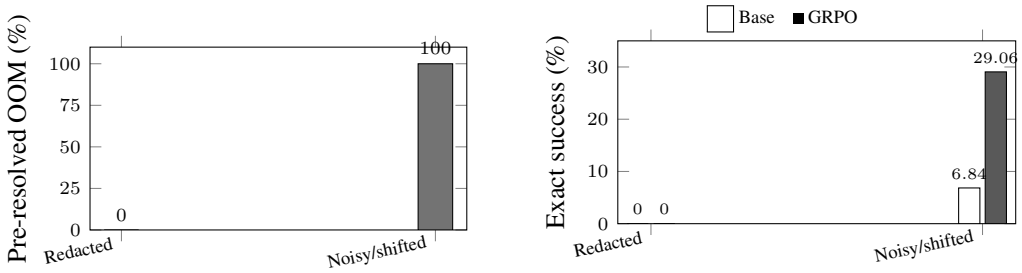

Figure 2: Profile-stratified audit of the 192 affected episodes. Left: the active OOM sub-fault is pre-resolved in every noisy or shifted-noisy episode and no redacted episode. Right: neither policy achieves exact success without that leaked state on this affected subset.

We reconstructed every affected event from its seed, profile, raw sandbox history, active sub-faults, and pre-policy state. If the injected history is harmless, every active sub-fault must be unresolved at x_{pre}. We joined the reconstruction with per-episode verifier outputs and report counts rather than a significance test because the defect has a deterministic state signature.

5 A HIDDEN PARTIAL-REMEDIATION DEFECT

The generator adds a decoy remediation for noisy and shifted-noisy profiles. Its literal-name special case is:

```
if active_fault_name == "oom_kill":  fix_dependency()
else:  restart_app()
```

The intent is to avoid giving away the fault through a prior successful repair. However, `restart_app()` sets the process-running condition used by the OOM-kill sub-fault verifier. The literal exception protects only the workflow named `oom_kill`; it does not protect composite workflows containing that same sub-fault.

Twelve composite workflows contain OOM kill as a sub-fault: `container_image_drift`, `cpu_throttling`, `dependency_mismatch`, `feature_flag_dependency_chain`, `file_descriptor_exhaustion`, `load_balancer_misroute`, `memory_pressure_high`, `read_write_split_misconfig`, `schema_migration_pending`, `transitive_dependency_conflict`, `upstream_health_check_failure`, and `zombie_process_accumulation`.

Table 1 is an exact state-level boundary, not a weak correlation. In every leaked event, the final history action is `restart_app()`, the process is already running, and the OOM-kill sub-fault is resolved before the policy acts. In every redacted event, the process is not running and that sub-fault remains unresolved. The retained exact-success rates in the leaked group are 6.84% for the base policy and 29.06% for GRPO; both policies have zero exact successes in the reconstructed redacted group.

The defect affects 34 of the retained GRPO policy's 228 exact successes (14.9%) and 8 of the base policy's 16 exact successes (50.0%). These fractions are not a valid correction to the headline result: the same generator path contaminates training exposure and changes the task distribution. A clean dataset, matched base evaluation, and fresh training run are necessary before making a comparative policy claim.

6 CORRECTED PROTOCOL

The repair is small but the validation obligation is broad. First, the generator should determine whether a candidate historical action changes *any* active sub-fault in the current workflow, rather than special-casing a workflow label. Second, it should assert the invariant after constructing every episode and fail closed on violation. Third, this assertion must be parameterized over every workflow, seed, and observation profile.

A practical regression suite should include:

1. an exhaustive generator test over workflows and profiles asserting zero pre-policy resolved active sub-faults;
2. an action-effect test rejecting decoy history actions whose transition intersects active verifier predicates;
3. versioned scenario manifests recording seed, profile, fault set, generator revision, and a hash of pre-policy state; and
4. evaluation reports with per-episode identifiers and verifier traces sufficient to reconstruct aggregate claims.

Future learning experiments should use only the repaired generator, freeze a versioned scenario manifest before fitting, evaluate the untouched base policy on the same manifest, and report exact success, partial reward, validity failures, and per-workflow outcomes.

7 DISCUSSION AND LIMITATIONS

This audit concerns a controlled infrastructure environment and one concrete defect. It does not imply that all agent benchmarks have state leakage, nor does it estimate either policy's clean performance. More subtle leaks may require counterfactual generation and blinded review in addition to state assertions.

The broader lesson is methodological. A terminal verifier answers whether the system is healthy now. A valid benchmark must also answer whether it was unhealthy in the relevant way when the agent began. Treating those as separate contracts gives benchmark authors a concrete failure mode to test and reviewers a concrete artifact to request.

ETHICS STATEMENT

The environment concerns operational remediation and should be used only in isolated sandboxes, never as authorization to execute actions against production systems. We report the retained benchmark defect and invalidate the affected policy comparison to avoid overstating system capability. The audit method, scenario reconstruction procedure, and generator-level integrity tests are intended to support reproducible correction of the benchmark.

REPRODUCIBILITY STATEMENT

The supplemental audit bundle contains the complete 52-workflow per-fault table, all 1,664 retained per-episode ordered action records, the 32 reconstructed observations used for the initial cross-fault inspection, source snapshots from dataset commit `7e2fd21a12710ee9c2f3661918b7b35c9e2d08b8`, run identifiers, and the held-out-file hash. Section 4 states the reconstruction procedure, and the appendix lists the state and provenance fields required to repeat it. The paper deliberately reports no corrected learning result: reproducing such a result requires regeneration and retraining after the integrity fix.

AI USE STATEMENT

In this work, we used OpenAI generative AI tools (ChatGPT and Codex) to assist with benchmark-code inspection, implementation and testing of audit utilities, refinement of the audit methodology,

reformatting and analysis of retained per-episode artifacts, checking aggregate calculations, and interpretation of debugging evidence. These tools also assisted with manuscript structure, drafting and editing, title formulation, literature suggestions, reference formatting, and LaTeX preparation. Generative AI was not used to generate the retained benchmark outcomes reported in this paper. It was not used for translation, qualitative or thematic analysis, or the formulation or proof of mathematical theorems; those tasks were not applicable. AI-assisted code and calculations were executed against the retained artifacts and source code, and all AI-assisted outputs were reviewed by the author. The author takes responsibility for the final content, including text, claims, code, and artifacts produced with the aid of generative AI.

REFERENCES

Carlos E. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations*, 2024.

Xiao Liu et al. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations*, 2024.

Shuyan Zhou et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations*, 2024.

A AUDIT CHECKLIST

For each event, retain workflow and active sub-fault identifiers; seed and profile; pre-policy history actions; a pre-policy state snapshot or hash; integrity-assertion result; policy actions; and final verifier trace. An aggregate success number without this provenance cannot distinguish policy repair from generator-induced repair.

B EVIDENCE ACCOUNTING

The affected set contains 12 composite workflows and 16 held-out variations per workflow, for 192 episodes per policy. Profile assignment yields 75 redacted and 117 noisy or shifted-noisy episodes. Seed replay found 0/75 and 117/117 pre-resolved OOM sub-faults, respectively. Joining the same row identifiers to the retained reports yields base exact-success counts of 0 and 8, and GRPO counts of 0 and 34. The numerator used in each percentage in Figure 2 is therefore an integer count shown in Table 1.