Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Mutation Testing of Task-Scoped State Oracles in Software-Agent Benchmarks: A Cross-Benchmark Empirical Study

Shengyao Sun

sthfornothing@sjtu.edu.cn

Shanghai Jiao Tong University

Research Article

Keywords: mutation testing, test oracle, software agents, benchmark evaluation, side effects, stateful systems

Posted Date: August 18th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10665114/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Mutation Testing of Task-Scoped State Oracles in Software-Agent Benchmarks: A Cross-Benchmark Empirical Study

Shengyao Sun[1*]

[1*]SJTU Paris Elite Institute of Technology (SPEIT), Shanghai Jiao Tong University, 800 Dongchuan Road, Minhang District, Shanghai, 200240, China.

Corresponding author(s). E-mail(s): sthfornothing@sjtu.edu.cn;

Abstract

Stateful software-agent benchmarks increasingly use executable evaluators to decide whether an agent produced the requested persistent effects. Yet a high task score is informative only if the evaluator rejects harmful extra effects without rejecting equivalent representations. We introduce a deterministic mutation-testing protocol for calibrating such task-scoped state oracles. Starting from known-valid executions, the protocol injects six classes of harmful state faults and three schema-certified benign transformations, then compares official scores with three controlled reference rules. Across frozen samples from τ^2-Bench, ToolSandbox, and AppWorld, 53 of 60 attempted task templates yielded 145 applicable mutations. Official evaluators rejected 100 of 110 harmful mutations (90.9%; Wilson 95% CI 84.1–95.0%) and preserved all 35 applicable JSON-key-order variants (95% CI 90.1–100%). All ten false negatives occurred in ToolSandbox: four same-record and six cross-namespace collateral changes across six templates. A post-hoc diagnostic reproduced every extra effect through its public tool; all ten still received score 1.0 with unchanged evaluation hashes. AppWorld contributed only five applicable harmful mutations, so the study does not support a broad benchmark ranking. The results motivate allowed-state-delta regression tests that verify required changes and reject unexpected persistent changes.

Keywords: mutation testing, test oracle, software agents, benchmark evaluation, side effects, stateful systems

1 Introduction

Software agents increasingly act through tools that change persistent state: they modify orders, contacts, settings, files, and other records. Benchmarks for these agents therefore need more than textual answer matching. AppWorld evaluates interactions with a simulated application ecosystem using state-based tests Trivedi et al. (2024); ToolSandbox evaluates stateful conversations through milestone and minefield matching Lu et al. (2024); and τ^2-Bench provides programmatically verifiable tasks in which both user and agent can act through tools Barres et al. (2025). In each case, the evaluator is executable software that operationalizes task success.

That evaluator is also a test oracle. The oracle problem asks how a test decides whether observed behavior is correct Barr et al. (2015). For an agent task, checking only that a requested target changed can be insufficient: an agent may make the required change and also modify an unrelated field, duplicate a record, or alter another application. Conversely, an evaluator that compares raw representations too strictly may penalize a semantically equivalent state. A trustworthy score therefore requires both sensitivity to task-relevant faults and specificity to certified benign representations.

Recent research has already shown that benchmark flaws, reward hacking, unsafe side effects, and false success deserve systematic study Wang et al. (2026); Zhong et al. (2026); Advani (2026); Zhou et al. (2026). Our contribution is narrower. We treat the official state evaluator itself as the software under test and apply deterministic, task-conditioned mutation analysis. The protocol begins with a benchmark-provided or mechanically validated successful final state, injects one predeclared state mutation when public evidence makes that mutation constructible, and re-runs the pinned official evaluator. Harmful mutants probe missing or collateral effects; certified-benign variants probe representational over-sensitivity. The experiment does not call an LLM and does not estimate how often real agents cause these faults.

We apply the protocol to fixed samples of 20 attempted templates from each of τ^2-Bench, ToolSandbox, and AppWorld. Of 540 attempted task–stratum cells, 145 mutations were applicable. Official evaluators rejected 100/110 harmful mutants and preserved 35/35 applicable JSON object-key reorderings. The ten missed harmful mutants were concentrated in six ToolSandbox templates and involved same-record or cross-namespace collateral state. We separately reproduced those effects by invoking the metadata-identified public tools after the known-valid trajectories. All ten effects remained evaluator-visible yet retained full official credit. Source inspection connects these observations to task evaluators whose milestone constraints do not impose a final-state completeness condition on the tested trailing effects.

The paper makes four contributions:

1. a task-scoped persistent-state fault model with six harmful mutation classes and three conservative, schema-certified benign classes;
2. a reproducible calibration protocol that reports applicability, harmful-mutant rejection, benign preservation, task-cluster uncertainty, and blinded generator quality control;

3. a cross-benchmark empirical audit over three pinned evaluator architectures, including complete accounting for inapplicable and failed cells; and
4. a public-tool reproduction and source-level diagnosis of ten ToolSandbox over-credit cases, together with an allowed-state-delta regression pattern for evaluator developers.

The contribution is not the general idea that benchmarks can contain flaws, that mutation testing is useful, or that successful agents can have side effects. It is the combination of task-conditioned persistent-state mutations, joint sensitivity/specificity calibration, and a common protocol applied to official stateful-agent evaluators.

2 Related Work

2.1 Test oracles and mutation testing

Mutation testing evaluates a test suite by injecting controlled faults and asking whether the suite distinguishes the mutants from correct behavior Jia and Harman (2011). It is particularly useful when conventional coverage does not establish that assertions can detect incorrect outcomes. Mutation-driven techniques have also been used to generate or strengthen test oracles Fraser and Zeller (2012). Empirical work has found a relationship between detecting conventional program mutants and detecting developer-fixed real faults, while also identifying uncoupled faults and operator limitations Just et al. (2014). Later guidance emphasizes explicit fault models, mutant selection, equivalent-mutant handling, and threats to validity when mutation scores are used in controlled experiments Papadakis et al. (2019). The broader oracle problem remains fundamental because executing a test does not itself determine whether the result is correct Barr et al. (2015).

Our study transfers this adequacy question from program test suites to benchmark evaluators. The mutant is not a source-code patch or a generated solution; it is a deterministic intervention on a known-valid persistent final state. The measured response is the score produced by the official task evaluator. SWE-Mutation also uses mutants to measure test adequacy, but its 2,636 program mutants assess LLM-generated software test suites rather than task-scoped persistent-state evaluators Sun et al. (2026). We therefore rely on established mutation-testing principles while changing the object, fault model, and evaluation context.

2.2 Stateful software-agent benchmarks

AppWorld supplies a controllable world of applications and evaluates task execution with state-based tests, including checks intended to detect unexpected changes Trivedi et al. (2024). ToolSandbox represents mutable device and application state as trajectories, matches declared milestones to snapshots, and can use minefields to penalize prohibited states Lu et al. (2024). τ^2-Bench studies conversational dual-control tasks with programmatically verifiable outcomes Barres et al. (2025). These designs make final or intermediate state first-class evaluation evidence, but they differ in what state is projected into a task oracle and how alternative successful trajectories are accepted.

We do not compare agent leaderboards or simulator realism. Instead, we pin one released implementation of each benchmark and ask whether its task evaluators respond to controlled state interventions. AppWorld's explicit unexpected-change checks serve as a positive design precedent; they are not treated as evidence that AppWorld is defect-free. Likewise, the present ToolSandbox result concerns six task templates in one pinned release, not the benchmark family as a whole.

2.3 Auditing benchmark evaluators and agent safety

AgentBoard supplements final success with progress metrics and multi-faceted analysis across multi-turn environments Ma et al. (2024), illustrating that a single end score can hide where an agent advanced or failed. ToolEmu scales agent-risk discovery using LM-emulated tools and an LM-based safety evaluator whose detected failures are checked by human judgment Ruan et al. (2024). These systems evaluate agent behavior and process signals; they do not mechanically calibrate the official persistent-state oracle against counterfactual final states.

BenchJack systematically audits reward-hacking flaws across ten agent benchmarks Wang et al. (2026), while adversarial hacker–fixer loops generate and repair verifier exploits over large terminal-agent task sets Zhong et al. (2026). These works establish that benchmark evaluators are attack surfaces and can benefit from adversarial hardening. Our protocol differs by using a small, deterministic persistent-state fault model rather than model-generated exploits, and by measuring both harmful detection and certified-benign preservation.

AgentS4D separates task completion from unsafe runtime behavior in workspace-agent executions Zhou et al. (2026). Work on false success studies cases in which trajectories or agent reports appear successful despite state-grounded task failure, including data from τ^2 and AppWorld Advani (2026). Those studies make the agent, trajectory, or safety harness the main empirical subject. Our main experiment instead holds the successful execution fixed and intervenes counterfactually on persistent state to isolate evaluator response. The perspectives are complementary: observational studies estimate failure behavior under agents, whereas evaluator mutation testing asks whether a specified fault would be visible to the scoring oracle at all.

3 Research Questions and Hypotheses

We ask four research questions:

RQ1: How often does an official evaluator reject a known-valid final state after one predeclared harmful task-effect mutation is injected?
RQ2: How does rejection sensitivity vary across required-effect omissions, wrong required values, duplicate effects, same-record collateral changes, same-namespace collateral changes, and cross-namespace collateral changes?
RQ3: Does the evaluator preserve its score under schema-certified equivalent representations?
RQ4: How do the official evaluators compare with global-exact, task-projected, and delta-complete reference rules?

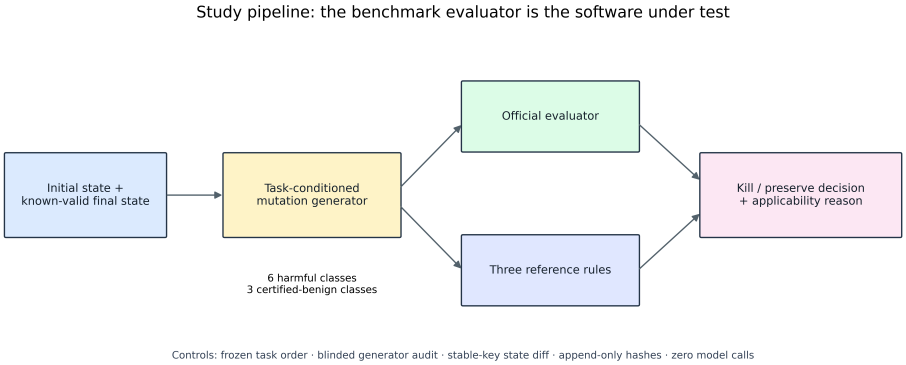

Fig. 1 Study pipeline. The benchmark evaluator is the software under test. Mutations are generated only when public task, schema, or tool evidence satisfies the frozen applicability rule.

The frozen hypotheses were: (H1) official evaluators would differ by at least 10 percentage points in harmful-mutant rejection for at least one stratum; (H2) task-projected evaluation would reject fewer cross-record or cross-namespace collateral mutations than delta-complete evaluation; and (H3) global-exact evaluation would preserve fewer certified-equivalent representations than task-projected or delta-complete evaluation. Unsupported and negative hypotheses were retained.

4 Method

4.1 Study design

Figure 1 shows the study pipeline. The unit under study is a benchmark evaluator, not an LLM or agent policy. Starting from a benchmark-provided or mechanically validated successful execution, the mutation generator creates at most one deterministic mutant for each applicable stratum. We then re-run both the official evaluator and three controlled reference rules. No model-generated judgment enters the labels or scores.

The original protocol was frozen on 1 August 2026 before formal scoring and initially targeted exhaustive eligible task catalogs. A later documented amendment restricted reported estimates to the 20 fixed preflight templates per benchmark after two additional τ^2 catalog-tail tasks exceeded the local runtime boundary. Runtime and applicability records were visible, but confirmatory estimates had not been interpreted. Tail rows remain append-only and are excluded from the present estimates. We therefore describe the work as a frozen-sample audit with a protocol deviation, not an exhaustive census.

4.2 Benchmarks, versions, and sample flow

We used τ^2 commit `fc0055dc4e0a316c3f83133267fbd6faaa770992`, ToolSandbox commit `165848b9a78cead7ca7fe7c89c688b58e6501219`, and AppWorld package `0.1.3.post1` with wheel SHA-256

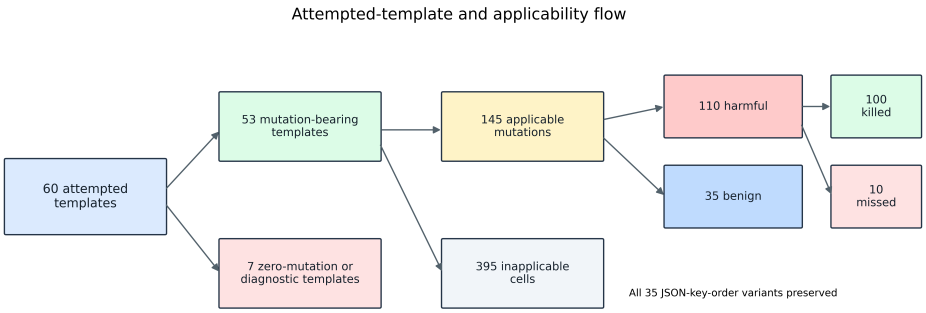

Fig. 2 Attempted-template and mutation-applicability flow. The denominator of evaluator-performance estimates is applicable mutations, not all 540 attempted cells.

Table 1 Frozen scope and applicable mutations.

Benchmark	Attempted	Bearing	Cells	Harmful	Benign
τ^2	20	20	180	35	20
ToolSandbox	20	18	180	70	0
AppWorld	20	15	180	5	15
Total	60	53	540	110	35

`db77f8003982502383a50fa2974983894bd1c54f64e2fd3f7e1540d5edd037eb`. The balanced scope attempted 60 task templates, nine task–stratum cells per template, and 540 cells in total.

Applicability was determined before treating an evaluator response as an outcome. A cell was applicable only if the mutation could be constructed from public schema or tool evidence and passed benchmark-specific shape and reachability checks. Every inapplicable cell retained a machine-readable reason. Figure 2 and Table 1 show the resulting sample flow.

Of 540 cells, 145 yielded applicable mutations and 395 were inapplicable. Fifty-three of 60 attempted templates contributed at least one mutation. One ToolSandbox template had a gold score below the frozen 0.95 gate and another supplied no certified candidate. Four AppWorld templates exposed no persistent state change and one encountered a retained Windows `UnicodeEncodeError` during gold execution. A post-hoc rerun under explicit UTF-8 settings completed that task with score 1.0 and 27 persistent change units; the historical exclusion was not replaced.

4.3 Task-conditioned state fault model

Table 2 defines the frozen mutation operators. To reduce researcher discretion, the generator selected the lexicographically first eligible field or record. Mutants requiring unreachable direct state corruption were excluded. Benign transformations required

Table 2 Frozen mutation operators and intended oracle obligation.

Class	Intervention	Expected evaluator behavior
Required revert	Restore one required changed field to its initial value	Reject
Required wrong value	Replace one required field with a schema-valid but task-wrong value	Reject
Duplicate effect	Duplicate a newly created record or persistent effect when multiplicity was not requested	Reject
Same-record collateral	Alter a non-required mutable field on the target record	Reject
Same-namespace collateral	Alter a reachable non-target record in the same namespace	Reject
Cross-namespace collateral	Alter a reachable record in another mutable namespace or application	Reject
JSON object-key order	Reorder keys where normalization certifies equivalence	Preserve
Numeric lexical equivalence	Change only a schema-certified numeric representation	Preserve
Set-order equivalence	Reorder a schema-certified unordered collection	Preserve

public schema or normalization evidence; free-form paraphrases and model-generated equivalence judgments were prohibited.

4.4 Official and reference evaluators

Let s_g be an evaluator's score for the known-valid final state and s_m its score for a mutant. A harmful mutant was killed when $s_m < s_g - 10^{-12}$; otherwise it was over-credited. A benign variant was preserved when $|s_m - s_g| \leq 10^{-12}$. The primary outcomes were harmful-kill rate and benign-preservation rate. The Scope Calibration Score (SCS), the harmonic mean of aggregate harmful-kill and benign-preservation rates, was secondary because an aggregate can conceal stratum-specific failures.

Three frozen reference rules were applied to the same synthetic states. *Global-exact* compared the complete normalized persistent state with the known-valid final state. *Task-projected* compared only the required target projection. *Delta-complete* required the intended changes and rejected unexpected persistent changes outside an allowed set. These are diagnostic rules, not drop-in replacements for benchmark semantics; global-exact can reject legitimate alternative solutions, while delta-complete requires a correctly specified allowed-effect set.

4.5 Blinding, generator audit, and integrity

A 30-item outcome-blinded audit validated initial required-revert and JSON-key-order candidates (30/30 valid). A supplementary stratified audit inspected 15 candidates from the other strata. Fourteen were valid and one, S004, revealed positional list alignment after a deletion. The invalid record was retained, the generator was changed to align records by stable primary key, and a separately blinded replacement R001 passed. The effective stratified gate was 15/15, but the historical defect remains disclosed.

The runners used deterministic seeds, append-only JSONL logs, stable cell identifiers, state and evaluator-output hashes, source-manifest hashes, and resumable execution. Replaying the frozen arm reproduced the scientific fingerprint for every benchmark. The confirmatory experiment used zero model/API calls and incurred zero paid API cost.

4.6 Statistical analysis

We report counts, percentages, and Wilson 95% confidence intervals for applicable mutations. Because one task could contribute several mutations, the task template was the dependence unit in an additional cluster bootstrap with 10,000 resamples and seed `20260801`. Paired evaluator-rule outcomes were compared using two-sided exact McNemar tests, paired percentage-point differences, and Holm correction within each research-question family. We distinguish observed effect thresholds from population inference and do not interpret score differences as evidence about model mechanisms.

4.7 AI-assisted development

LLM-based generative-AI assistants, including OpenAI ChatGPT/Codex, were used during software and manuscript development to suggest code scaffolds and debugging steps and to assist prose drafting and revision. The author selected, inspected, executed, and validated all resulting code and text. No LLM generated confirmatory labels, judged evaluator outcomes, produced experimental responses in the confirmatory arm, or supplied statistical data. All confirmatory results were computed by deterministic scripts from archived evaluator outputs, and the author verified the source interpretations, claims, and final manuscript. The separately reported stopped exploratory model arm in Appendix A used model APIs as an object of study and is excluded from confirmatory claims.

5 Results

5.1 RQ1: Aggregate harmful-mutant rejection

Across 110 applicable harmful mutations, official evaluators rejected 100 (90.9%; Wilson 95% CI 84.1–95.0%; task-cluster bootstrap interval 84.0–97.3%). Table 3 gives benchmark-specific counts. AppWorld's 5/5 result has a lower Wilson bound of 56.6% and covers only required-effect faults; it is not evidence of complete collateral-effect detection.

Table 3 Official evaluator performance on applicable mutations.

Benchmark	Harmful killed	Total	Rate	Wilson 95% CI	Benign preserved	Total	Rate
τ^2	35	35	100.0%	90.1–100.0%	20	20	100.0%
ToolSandbox	60	70	85.7%	75.7–92.1%	0	0	–
AppWorld	5	5	100.0%	56.6–100.0%	15	15	100.0%
Overall	100	110	90.9%	84.1–95.0%	35	35	100.0%

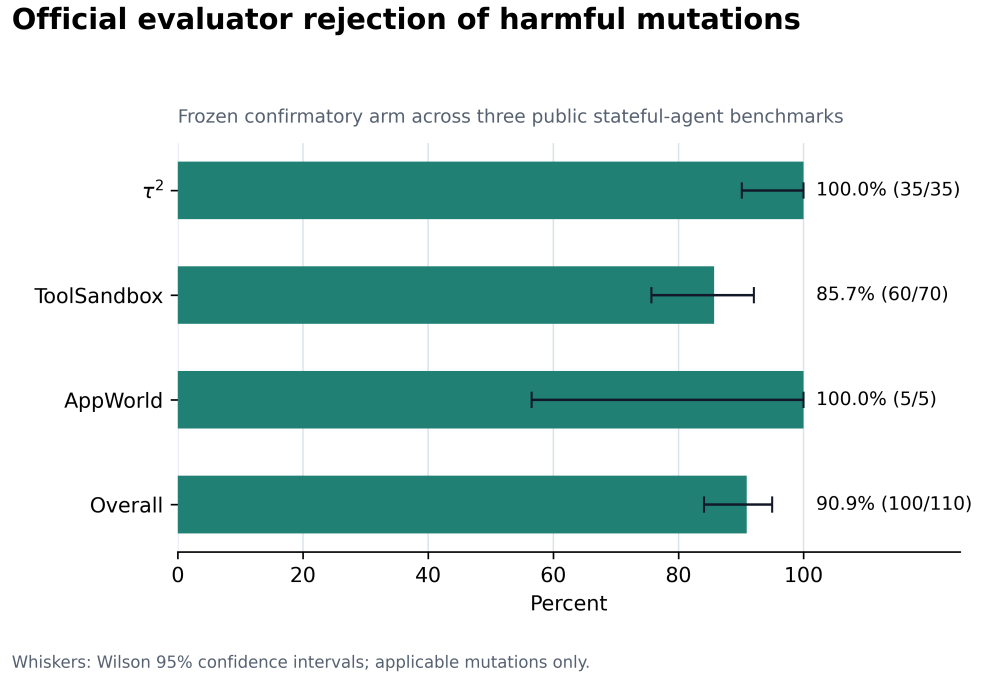

Fig. 3 Official harmful-mutant rejection with Wilson 95% confidence intervals. Counts and applicability must be considered alongside rates.

5.2 RQ2 and H1: Sensitivity by fault class

Required-effect faults were consistently detected in the observed sample: official evaluators rejected all 20 required reverts and all 30 wrong-value mutations. They also rejected all 14 duplicate effects and all 18 same-namespace collateral mutations. Misses were concentrated in ToolSandbox, which rejected 3/7 same-record collateral mutations (42.9%; Wilson 15.8–75.0%) and 12/18 cross-namespace collateral mutations (66.7%; 43.7–83.7%).

The largest observed cross-benchmark stratum gap was 57.1 percentage points for same-record collateral effects: τ^2 rejected 3/3 and ToolSandbox rejected 3/7. This exceeds H1's frozen 10-point threshold and gives observed-effect support. The denominators are too small for a stable population ranking, so we do not claim that one benchmark architecture is generally more sensitive than another.

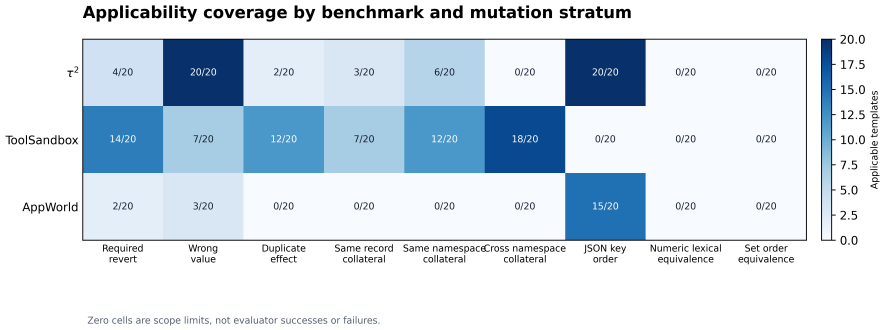

Fig. 4 Applicability and official response by benchmark and mutation class. Empty strata are scope limits, not evaluator passes.

5.3 RQ3 and H3: Certified-benign preservation

Official evaluators preserved all 35 applicable benign variants (100%; Wilson 90.1–100%): 20 τ^2 and 15 AppWorld JSON object-key reorderings. ToolSandbox supplied no applicable benign variants under the frozen generators. All reference rules also preserved 35/35. H3 is not supported. More importantly, numeric lexical and set-order equivalence had zero applicable cases. The result therefore concerns normalized JSON key order only and is not a broad specificity guarantee.

5.4 RQ4 and H2: Reference-rule comparison

Global-exact and delta-complete each rejected 110/110 harmful mutants; the official evaluators rejected 100/110; task-projected rejected 50/110. Relative to delta-complete, the official rate was 9.1 percentage points lower (task-cluster bootstrap interval -16.0 to -2.8 points; Holm-adjusted exact McNemar $p = 0.0039$). Official evaluators exceeded task-projected by 45.5 points (bootstrap 34.8–54.7; adjusted $p = 5.33 \times 10^{-15}$).

For the 46 applicable same-record, same-namespace, and cross-namespace collateral mutations, task-projected rejected 0/46 while delta-complete rejected 46/46 (adjusted $p = 2.84 \times 10^{-14}$). H2 is supported, but this contrast largely validates the reference-rule definitions: projection intentionally ignores out-of-projection changes, whereas delta completeness intentionally rejects them. It is not a headline empirical discovery.

Across all 145 applicable cases, official evaluators classified 135 correctly (93.1%), global-exact and delta-complete 145 (100%), and task-projected 85 (58.6%). The pooled rates depend on the observed class mixture, and the perfect reference scores have a construct advantage because the rules operate on the same controlled representation used to generate the mutants.

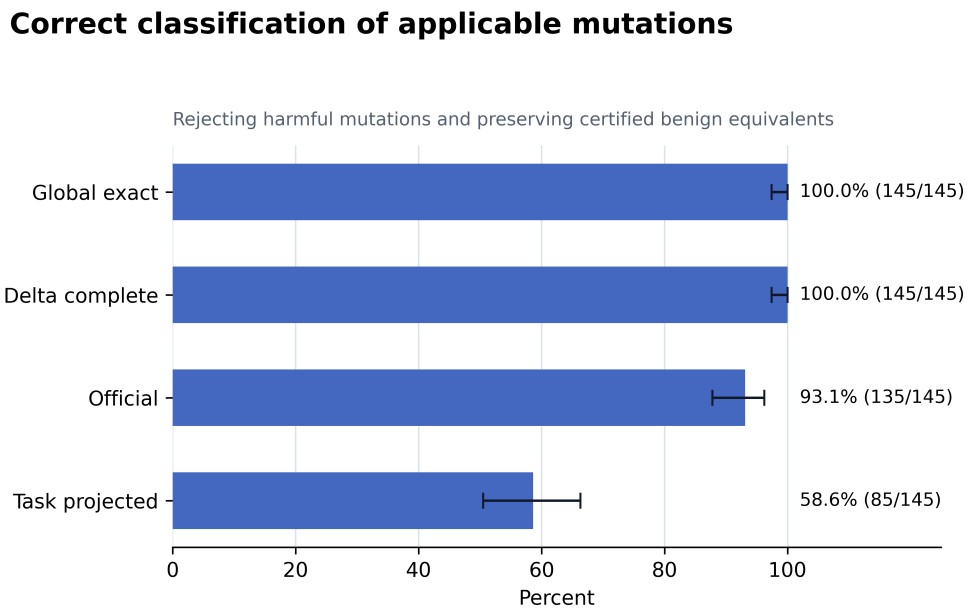

Fig. 5 Classification of applicable harmful and certified-benign mutants by official and reference rules. Reference rules are controlled diagnostics, not replacement evaluators.

6 Error Analysis

All ten official false negatives occurred in ToolSandbox and were spread across six of its 20 attempted templates. Four were same-record collateral changes and six were cross-namespace changes. Target namespaces were CONTACT (four), REMINDER (four), and SETTING (two). Every mutant retained gold score 1.0 and the same evaluation-output hash as the known-valid trajectory. Global-exact and delta-complete rejected all ten; task-projected accepted all ten.

The frozen state reconstruction could backdate a mutation when the target namespace was untouched in the gold trajectory. We therefore ran a post-hoc diagnostic, excluded from confirmatory statistics. For each missed case, it invoked the metadata-identified public tool in the gold execution context, advanced the trajectory to expose the resulting trailing state, checked the target value and fault classification, and reran the official evaluator. All 10/10 tool calls succeeded and retained their class. Six changed one persistent field; four reminder calls changed two because the public tool also refreshed a timestamp. The official score and evaluation hash remained unchanged in all ten. This falsifies failed public reachability and snapshot-replacement timing as explanations for the over-credit, but it remains post-hoc evidence.

Source inspection of the pinned ToolSandbox release provides an evaluator-level mechanism consistent with the observations. Its scorer searches trajectory snapshots for the mapping that maximizes declared milestone similarity and can nullify that

similarity if a minefield matches Lu et al. (2024). None of the six affected task templates declares a minefield. In same-record cases, a milestone constrains selected target columns, leaving the added field outside that constraint. In cross-namespace cases, required milestones can be matched before an appended side-effect snapshot. The tested task evaluators therefore contain no final-state completeness condition that requires the final persistent delta to stay within an allowed set.

This explanation is deliberately narrow. It applies to six templates and the pinned source version. It does not establish the prevalence of such faults across ToolSandbox, later releases, or real-agent trajectories.

7 Discussion

7.1 What the findings establish

The experiment establishes a causal property of the tested software oracles: for each of ten reachable ToolSandbox state interventions, the pinned official evaluation output did not change even though a non-required persistent effect was added to a valid execution. Because the intervention was deterministic and the valid trajectory was held fixed, the result is stronger than observing that a particular agent failed. The public-tool diagnostic further shows that the effects can be produced through benchmark-exposed operations and remain visible in a trailing state.

The experiment does not establish how frequently real agents produce such effects, whether users would assign equal harm to every intervention, or whether the same omissions appear in other task templates. The stopped exploratory model arm in Appendix A illustrates why these are separate questions: its early episodes mainly measured whether heterogeneous agents completed benchmark interactions, adding provider and interface confounds without strengthening the evaluator-level causal claim.

7.2 Applicability is part of the result

Only 145/540 task–stratum cells were applicable, and the distribution was uneven. Rather than treating this as missing data that can be ignored, the protocol reports it as information about each benchmark's public state and tool surface. AppWorld contributed only required-effect harmful mutants in the balanced sample, so no conclusion about its handling of collateral state is warranted. ToolSandbox contributed no schema-certified benign variant. τ^2 supplied no cross-namespace mutant. A benchmark comparison that pooled zero-applicability cells as passes would inflate confidence and obscure these architectural differences.

Future calibration suites should therefore begin with a benchmark-specific applicability map. Shared fault-class names do not guarantee shared constructibility. Reporting attempted cells, mutation-bearing templates, and applicable mutations prevents a broad-looking design from being mistaken for balanced empirical coverage.

7.3 An allowed-state-delta regression pattern

The practical implication is an additional regression obligation, not wholesale replacement of task semantics. Let S_0 be initial persistent state, S_f the observed final state, R the set of required path-value changes, and A an explicitly justified set of additional allowed changes. An evaluator-side regression oracle can require

$$R \subseteq \Delta(S_0, S_f) \quad \text{and} \quad \Delta(S_0, S_f) \setminus (R \cup A) = \emptyset, \tag{1}$$

where Δ aligns records by stable primary key before comparing fields. Positive tests should include valid alternative solutions and certified normalizations. Negative tests should inject each reachable collateral mutation and assert that the evaluator score falls or a minefield/guardrail fires. This pattern complements milestone matching: milestones show that requested effects occurred, whereas an allowed-delta check tests that no unlisted persistent effect survived.

The artifact includes a small reference implementation and five regression tests. It handles stable-key alignment, required target values, explicit prefix allowlists, and rejection of same-record, same-namespace, and cross-namespace extras. It is a design template rather than an upstream ToolSandbox patch. Real benchmarks must specify legitimate incidental changes such as timestamps, generated identifiers, caches, and alternative valid task solutions. An empty allowlist can be as misleading as an incomplete evaluator.

7.4 Implications for benchmark maintainers and users

For maintainers, state mutation tests can be added to continuous integration whenever a task or evaluator changes. A task should not be released only with one successful trajectory; it should include negative oracles that reverse a required effect, change a required value, duplicate multiplicity, and inject reachable collateral state. Certified-benign transformations should also be tested so stricter checks do not collapse legitimate equivalence.

For benchmark users, a scalar task score should be interpreted as coverage of the evaluator's declared conditions, not a proof that the final environment is globally safe. Benchmark reports can improve transparency by publishing the namespaces and fields observed by each task evaluator, the role of guardrails or minefields, and known allowed side effects. For researchers comparing agents, evaluator calibration bounds what conclusions a leaderboard difference can support.

8 Threats to Validity and Limitations

Post-freeze scope amendment. The initial protocol targeted exhaustive eligible catalogs, but runtime limits led to a documented 20-attempted-template-per-benchmark arm. Although these template sets were fixed in preflight and tail records were retained, the change is a protocol deviation. The study is a frozen-sample audit, not an exhaustive census.

Small and uneven strata. Only 53/60 attempted templates yielded a mutation. AppWorld contributed five harmful mutations and no collateral mutation; τ^2 contributed three same-record and no cross-namespace mutation. Perfect rates have wide intervals. H1 is an observed practical signal rather than a population ranking.

Restricted benign coverage. All 35 applicable benign variants were normalized JSON key reorderings. Numeric lexical and set-order equivalence had zero applicable instances. Consequently, the preservation result cannot be generalized to benign representation changes broadly.

Synthetic external validity. Deterministic mutants isolate evaluator response but do not estimate real-agent fault prevalence, user harm, or production safety. The public-tool diagnostic establishes reachability for ten cases, not natural frequency.

Reference-rule construct advantage. Global-exact and delta-complete were defined over the controlled state representation used to generate mutants. They may reject legitimate alternative solutions or require unavailable state access. Their perfect result is calibration evidence, not proof of practical superiority.

Generator validity. Blinded audits caught and corrected one real stable-key alignment defect but sampled rather than manually inspected all 145 mutations. Deterministic selection, shape checks, public evidence, and regression tests reduce but do not eliminate label risk.

Implementation and platform. Results concern pinned versions executed on Windows. Later releases, different task samples, or evaluator configurations may differ. One historical AppWorld encoding exclusion did not reproduce under explicit UTF-8 settings but remains excluded by the frozen rule.

Statistical dependence and multiplicity. Several mutations can arise from one task. We supplement row-level Wilson intervals with task-cluster bootstrap intervals, but sparse strata remain imprecise. H2 follows the definitions of the reference rules and should not be over-interpreted based on its small p-value.

9 Protocol Deviations and Transparency

Table 4 summarizes decisions that could otherwise be mistaken for hidden researcher flexibility. All superseded and failed records remain in the artifact.

10 Conclusion

Task success in a stateful software-agent benchmark is a claim made by executable oracle code. We calibrated that claim using deterministic task-scoped state mutations across fixed samples of three benchmarks. Official evaluators rejected 100/110 harmful mutants and preserved 35/35 applicable JSON-key-order variants, but ten reachable collateral effects across six ToolSandbox templates retained full credit. The finding is neither a benchmark-wide safety estimate nor an agent failure rate. It is evidence that these pinned task evaluators lacked a final-state completeness condition for the tested effects. Mutation tests that pair required-effect assertions with explicit allowed-state-delta checks offer a practical regression pattern. Equally important, future studies must report where mutations are inapplicable and must test benign representations, so evaluator sensitivity is not purchased by rejecting valid alternatives.

Table 4 Protocol deviations and reporting consequences.

Stage	Trigger	Outcomes visible?	Action	Consequence
Confirmatory scope	Two τ^2 tail tasks exceeded runtime boundary	Runtime only	Report fixed 20-template preflight set per benchmark	Frozen-sample audit, not exhaustive census
Generator audit	S004 used positional alignment after deletion	Labels only	Stable-key correction and blinded R001 replacement	S004 excluded and retained historically
AppWorld diagnostic	Windows encoding exception	Yes	Post-hoc UTF-8 rerun only	Frozen denominator unchanged
Provider substitution	Kimi allocation unavailable	No	Replaced six exploratory cells before live outcomes	No effect on confirmatory arm
Exploratory model arm	Seven cells had low decision value	Yes	Stopped remaining 29 cells	Inclusion gate failed; excluded from claims

Data Availability Statement

The key-free replication package is supplied with this submission as Online Resource 1. It contains the frozen protocol and amendments, pinned source identifiers, deterministic mutation generators, applicability reasons, append-only result logs, validators, analysis code, derived tables and figures, aggregate blind-audit decisions, the historical invalid audit record and replacement, and scientific fingerprints. Upstream benchmark material is redistributed only where its license permits; otherwise, scripts and hashes reconstruct the analysis from public packages. The confirmatory and replication analyses require no API key. A persistent repository DOI may be added in a revision without changing the archived scientific results.

Declarations

Author contributions. Shengyao Sun: conceptualization, methodology, software, formal analysis, investigation, data curation, visualization, writing—original draft, and writing—review and editing.

Funding. This research received no external funding.

Competing interests. The author declares no financial or non-financial competing interests.

Ethics approval. The study used public software artifacts and synthetic benchmark states and did not involve human participants or personal data.

Generative-AI use. As detailed in Section 4.7, generative AI assisted with code and prose drafting under author supervision. It did not label confirmatory mutations, judge evaluator outcomes, generate experimental responses, or supply statistical data.

All numerical results were computed by deterministic scripts from archived evaluator outputs. The author remains responsible for source verification, claims, and the submitted manuscript.

A Stopped Exploratory Model Arm

A frozen 36-episode secondary arm was designed to examine whether generated trajectories naturally reached a conflict state in which the official evaluator accepted but delta-complete rejected. It was terminated outcome-unblinded after seven episodes because it mainly measured heterogeneous agents' task completion and interface adherence and had not reached the predeclared 24/36 manuscript-inclusion threshold. Four τ^2 and three AppWorld episodes completed; no ToolSandbox episode completed. Three episodes received official acceptance and exactly matched the known-valid persistent state. Four were rejected and made no persistent change. No harmful side effect or evaluator conflict was observed, but the Wilson upper bound for 0/7 is 35.4%, so the data do not support a zero-rate claim. The arm consumed 355,381 agent-plus-simulator tokens and 100 generation calls. It is excluded from every confirmatory estimate and provider comparison. The public artifact retains the aggregate stopped-arm summary and termination amendment but excludes raw provider transcripts, which contain synthetic account records and are unnecessary for reproducing the confirmatory claims.

References

Advani L (2026) From confident closing to silent failure: Characterizing false success in LLM agents. URL https://arxiv.org/abs/2606.09863, accepted to FAGEN at ICML 2026, arXiv:2606.09863

Barr ET, Harman M, McMinn P, et al (2015) The oracle problem in software testing: A survey. IEEE Transactions on Software Engineering 41(5):507–525. https://doi.org/10.1109/TSE.2014.2372785

Barres V, Dong H, Ray S, et al (2025) τ^2-bench: Evaluating conversational agents in a dual-control environment. URL https://arxiv.org/abs/2506.07982, arXiv:2506.07982

Fraser G, Zeller A (2012) Mutation-driven generation of unit tests and oracles. IEEE Transactions on Software Engineering 38(2):278–292. https://doi.org/10.1109/TSE.2011.93

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

Just R, Jalali D, Inozemtseva L, et al (2014) Are mutants a valid substitute for real faults in software testing? In: Proceedings of the 22nd ACM SIGSOFT International

Symposium on Foundations of Software Engineering. Association for Computing Machinery, pp 654–665, https://doi.org/10.1145/2635868.2635929

Lu J, Holleis T, Zhang Y, et al (2024) Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. URL https://arxiv.org/abs/2408.04682, arXiv:2408.04682

Ma C, Zhang J, Zhu Z, et al (2024) AgentBoard: An analytical evaluation board of multi-turn LLM agents. In: Advances in Neural Information Processing Systems, https://doi.org/10.52202/079017-2365, URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html

Papadakis M, Kintis M, Zhang J, et al (2019) Mutation testing advances: An analysis and survey. In: Advances in Computers, vol 112. Elsevier, p 275–378, https://doi.org/10.1016/bs.adcom.2018.03.015

Ruan Y, Dong H, Wang A, et al (2024) Identifying the risks of LM agents with an LM-emulated sandbox. In: International Conference on Learning Representations, URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/7274ed909a312d4d869cc328ad1c5f04-Abstract-Conference.html

Sun Y, Zhao Y, Wang Y, et al (2026) SWE-mutation: Can LLMs generate reliable test suites in software engineering? In: Findings of the Association for Computational Linguistics: ACL 2026. Association for Computational Linguistics, pp 39651–39674, https://doi.org/10.18653/v1/2026.findings-acl.1976, URL https://aclanthology.org/2026.findings-acl.1976/

Trivedi H, Khot T, Hartmann M, et al (2024) Appworld: A controllable world of apps and people for benchmarking interactive coding agents. URL https://arxiv.org/abs/2407.18901, aCL 2024 camera-ready version, arXiv:2407.18901

Wang H, Li H, Mang Q, et al (2026) Do androids dream of breaking the game? systematically auditing AI agent benchmarks with BenchJack. URL https://arxiv.org/abs/2605.12673, arXiv:2605.12673

Zhong Z, Segal I, Bercovich I, et al (2026) Hardening agent benchmarks with adversarial hacker–fixer loops. URL https://arxiv.org/abs/2606.08960, arXiv:2606.08960

Zhou J, Ke Z, Ye J, et al (2026) AgentS4D: Benchmarking runtime risks across the execution lifecycle of LLM-based workspace agents. URL https://arxiv.org/abs/2607.27294, arXiv:2607.27294

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SQJReplicationArtifactv1.zip