

Agent evaluation harnesses are software: they run a test suite, drive a test input generator, and decide outcomes with a *test oracle*. This paper audits that software for two classes of quality defect that a score table cannot show.

First, we give a **systematic oracle-integrity auditing approach** — five state-integrity invariants and three known-answer controls — and *measure* its sensitivity and specificity on the implementations studied, by injecting seven oracle defect classes across two task domains and a synthetic domain. Every injected class is detected, and the repaired oracle the package ships yields no false positive. We do not claim a general-purpose framework: it was evaluated on the implementations reported here, and Section 6 reports where it does not transfer. Applied to a widely used agent benchmark, the suite exposes a defect in which the reward function resets the benchmark database to build a ground-truth hash and never restores it, so a harness that scores an episode *after* it has run reads the reference solution’s state: across 4,320 episodes the recorded labels report 4,194 database passes, an independent replay of the recorded tool calls gives 2,883, and measured success falls from 94.3% to 65.5%.

Second — and at weaker evidential strength, as an analysis of the measurement process rather than of any model — we show that localising the apparatus is a reliability property of the measurement. A per-component ablation over 3,240 episodes finds that all three documents the agent reads — its policy, its tool descriptions, and the strings its tools return on error — carry the effect, with tool descriptions largest (−9.2 pp) and error strings smallest (−5.3 pp) in an exploratory ordering. A second task domain reverses the sign of the language effect, so we bound the first finding to the configuration in which it was measured.

Third, we report where the auditing approach does *not* transfer. Applied to a second, independently implemented agent benchmark, it exposed a **portability constraint** rather than a validation: the reference terminal state a positive control needs cannot be reconstructed by an external harness from the framework’s recorded API-call log. We publish that attempt as a negative result — it bounds the claim to the implementations studied and identifies the contract an evaluation framework must expose before this auditing can be ported to it.

The paper is accompanied by a **replication package**: the auditing suite, the fault-injection catalogue, the pre-registration and deviation logs, the frozen per-episode records, and a single entry point that recomputes the offline analyses without network access, API keys or model calls.

When Agent Evaluation Becomes Unreliable: An Empirical Study of Oracle Failures and Configuration-Dependent Measurement in an LLM Agent Benchmark

Junjie Chen^{1*}

^{1*}School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China.

Corresponding author(s). E-mail(s): junjie_chen@njust.edu.cn;

Keywords: test oracle, flaky tests, measurement validity, benchmark quality, replication package, LLM agent evaluation

1 Introduction

Evaluating an AI agent means running it against a benchmark and reporting the number that comes out. That number — a success rate — is what a paper states, what a leaderboard ranks, and, increasingly, what a deployment decision rests on. It is not produced by the agent alone. It is produced by an *evaluation harness*: a suite of test cases (the benchmark’s tasks), a generator of test inputs (increasingly a language-model user simulator), and a *test oracle* — the benchmark’s reward function — that decides whether each run passed. The harness is software, and the score is one of its outputs (Figure 1).

Software engineering has spent decades on the failure modes of exactly this arrangement. Deciding expected from actual is the *oracle problem*, and it is where test suites fail most quietly (Barr et al, 2015). A test that passes and fails on unchanged code is *flaky* (Luo et al, 2014). A measurement that moves when the apparatus changes, while the property being measured does not, is a validity failure rather than a capability difference (Kitchenham et al, 2002). Fault injection, in the lineage of mutation testing (Jia and Harman, 2011), is the standard way to ask whether a test suite

would notice a defect at all. These are settled methods with an unsettled application area.

Agent benchmarks have adopted the software without adopting the methods. In particular, a benchmark’s reward function is treated as *definitional*: it defines what success means, so it is not itself tested. Two consequences follow. First, an error inside the oracle is invisible by construction, because the oracle is the instrument that would have detected it, and because the returned value can be correct — the defect can be in what reading the oracle *leaves behind*. Second, because the harness is also how a configuration is expressed (which language the simulator speaks, what the agent’s policy and tool descriptions say), a change in the apparatus is read as a change in the agent. Neither consequence is hypothetical: this paper reports one instance of each, found by auditing one widely used agent benchmark.

We therefore treat the evaluation harness as a system under test and audit it. We organise the work as three questions, of which only the first is about the evaluation system itself; the other two are analyses of the measurement process, and we state them at that strength.

RQ1 (primary). How reliable is the evaluation oracle — would it notice a defect in itself, and does it have one? The object of study is the evaluation system: the correctness and consistency of its verdicts and the fidelity of the labels it produces. The evidence is a test suite aimed at the oracle and a fault-injection battery aimed at the harness.

RQ2 and RQ3 (secondary). How do harness configurations influence observed evaluation outcomes, and how sensitive are the conclusions to factors that vary during execution? These are analyses of the *measurement process*, run on the same audited pipeline. Their object is not the model: we do not set out to characterise intrinsic language-model capability or behaviour. What they characterise is how design choices in the harness — which documents are localised, which task domain is used — expose, amplify or obscure variability that is present while a benchmark runs, and where the resulting conclusions stop holding.

We hold this distinction throughout: *evaluation-system reliability* is not *intrinsic model behaviour*. RQ1 provides direct evidence of the first — a defect in the oracle itself, found by tests aimed at the oracle. RQ2 and RQ3 provide evidence about the conditions under which a measurement outcome moves, and are reported as observations and boundaries rather than as findings about models. The three are not of equal evidential weight, and we do not present them as such.

Contributions.

1. **A systematic oracle-integrity auditing approach, with sensitivity and specificity measured on the implementations studied.** Eight checks in two families — five state-integrity invariants and three known-answer controls — plus a fault-injection battery of seven defect classes and a ground-truth-controlled synthetic domain. Validated across three targets; every injected defect class is detected, with no false positive on the repaired oracle the package ships, and each class maps to a minimal detector (Section 3.2).

2. **An oracle defect found by that method, a minimal reproduction, and a replay-based correction.** The benchmark’s `Env.calculate_reward()` resets the environment database to build a ground-truth hash and never restores it. The value it *returns* is correct: it is computed from a hash taken before the reset, and the benchmark’s own agent consumes that value without reading the environment again, so the benchmark’s own runner is unaffected. It becomes a wrong label only because our harness scores the episode *separately*, after the session has ended, and therefore re-reads an environment left holding the reference solution. We report this as an integration failure and assign it as one — the unsafe side effect is upstream, the second reading is ours — and make no claim that the upstream runner inflates its own success rates. On 4,320 episodes the archived labels record 4,194 database passes; replaying the recorded tool calls against a clean database gives 2,883, and success falls from 94.3% to 65.5% (Section 4.1).
3. **A configuration-dependent reliability effect, localised to its components.** Under correctly scored conditions, changing the simulator’s language yields no resolved difference in aggregate success, but it changes by a factor of two to ten the rate at which one agent’s trajectory degenerates. Localising the agent’s own world — its policy, its tool descriptions and its error strings — is the larger lever, and a per-component ablation over 3,240 episodes shows that all three documents carry a protective effect. The three effects do not add: localising the tool descriptions alone is measured as beating localising all three, by 1.11 pp. We report that ordering as an *observation* rather than a finding — the arms come from different run periods, the contrast is the most exposed of the ones we report, and our own power analysis puts a contrast of that size at 17% power in this design (Section 4.2). The sign of the effect does not survive a change of task domain, which bounds it to the configuration in which it was measured (Section 4.2).
4. **A replication package and the portability boundary of the method.** We release the auditing suite, the seven-class fault-injection catalogue, the pre-registration and deviation logs, and the frozen per-episode records behind every number in this paper, together with a single offline entry point that recomputes the analyses with no network access, no API keys and no model calls (Section 3.6). We also report, as a negative result, the second independently implemented benchmark on which the method did *not* transfer, and identify the framework contract that blocks it (Section 6).

What we do not claim.

The audit was completed on one agent benchmark, τ -bench (Sierra Research, 2024), with a second task domain, a synthetic domain, and one further, independently implemented benchmark on which it did not transfer. We therefore claim a *systematic auditing approach evaluated on the implementations studied*, not a general-purpose framework, and we report the portability boundary as a result rather than disposing of it. All limits are stated in Section 6.

2 Background and Related Work

2.1 Test oracles, mutation testing and the oracle problem

Deciding expected from actual is the oracle problem, a foundational difficulty of software testing (Barr et al, 2015). The trouble is not that oracles are absent but that they are *unexamined*: an oracle is exercised by every test run and tested by none, so its own faults are silent. Mutation testing grew out of this observation (Jia and Harman, 2011) — inject a fault, then ask whether the suite notices — and remains the standard technique for measuring what a test suite can actually detect. We take that technique and point it at the oracle itself rather than at the program under test.

Two properties of an oracle make our variant distinct. First, it can be *stateful*: an oracle that re-derives its expectation by replaying a reference solution against a live database can return a correct value and still leave the system in a state no other reader expects. Second, its contract with the harness is usually implicit — who reads the environment, and when, is not specified anywhere — so a second reader can be wrong through no fault of the oracle’s own runner. Our first contribution is a state-integrity formulation of the oracle problem for machine-learning evaluation harnesses, together with the fault-injection battery that measures it.

2.2 Flaky tests, measurement validity and configuration sensitivity

A flaky test passes and fails on the same code (Luo et al, 2014). In empirical software engineering the same worry appears as measurement validity: a reading that responds to the apparatus rather than to the property under study is not a weaker measurement of that property, it is a measurement of something else (Kitchenham et al, 2002). Metamorphic testing approaches the difficulty from a third direction, checking relations that must hold between related inputs when a direct oracle is unavailable (Chen et al, 2018).

Our second contribution sits in the middle of these three. The relation we test is a *measurement property*: changing the language of the apparatus, with task, agent, simulator model and repeat held fixed, should not change the measured ability. When it does, the honest reading is not that the agent got worse in one language but that the harness has a configuration sensitivity. In agent evaluation the analogous instability also has a level that CI flakiness does not: the *trajectory*, where a model’s multi-step behaviour degenerates non-deterministically rather than the same test returning two verdicts.

2.3 Testing and evaluating LLM-based systems

A separate strand treats a language model as the system under test, asking how to specify, exercise and judge a system whose outputs are not deterministic (Hou et al, 2024), (Zhang et al, 2022). We take that framing and apply it one level down. The system under test in this paper is not the model but the *evaluation harness*: the model is the input generator and the agent is the system the harness measures. The distinction matters because the two are routinely conflated — a benchmark’s reward

function is treated as definitional rather than as software with failure modes — and because a fault in the harness is invisible in precisely the way a fault in an oracle is.

2.4 Agent benchmarks and the reliability of their measurements

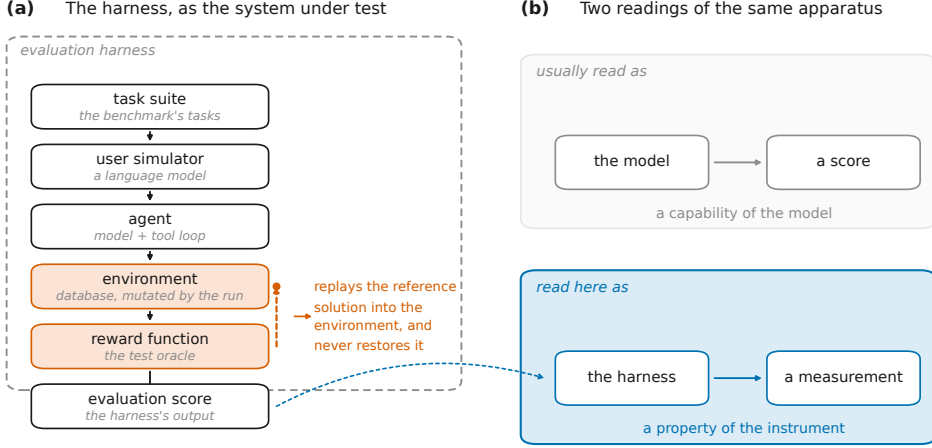
Benchmarks are themselves software artefacts whose validity and reproducibility must be argued rather than assumed (Recht et al, 2019), (von Kistowski et al, 2015). In agent evaluation the argument has mostly proceeded by measuring the measurement’s noise rather than by auditing its apparatus: model-as-judge scores vary with the judge and the prompt (Yin, 2026), (Mahmood et al, 2026); rankings shift under resampling or a change of reported statistic (Zheng and Yang, 2026), (Khosravi and Huo, 2026), (Bjarnason et al, 2026); aggregate success hides what actually happened in a run (Khanal et al, 2026); benchmarks can be answered without solving the task, or fail to discriminate between systems that differ (Kehkashan et al, 2026), (Bellibatlu et al, 2026); and the apparatus itself revisits established benchmarks (Yehudai et al, 2026), (Sierra Research, 2024), (Barres et al, 2026). We share that concern and change its object: instead of asking whether the score is noisy, we ask whether the instrument that produces it is *correct*, using tests aimed at the instrument.

2.5 User simulators as the measuring instrument

Because the user’s turns are generated rather than fixed, a user simulator is not a fixture of the benchmark so much as part of the measuring apparatus, and its own behaviour affects the reading (Seshadri et al, 2026), (Zhou et al, 2026), (Ni et al, 2026). Simulators are designed, validated and compared as components (Dou et al, 2025), (Jain and Sandhu, 2026), (Lee et al, 2026), (Ma et al, 2026), (Sekulić et al, 2024), (UserSim Organizers, 2026), and increasingly built on real interaction data (Zhou et al, 2023). A related line shows that the simulator’s presentation — persona, dialect, and the demographic cues it discloses — moves measured outcomes without moving model capability (Taday et al, 2026), (Haq and Saldías, 2026), (Tonneau et al, 2026), (Liu and He, 2026), and that the *language* of the apparatus is a configuration of this kind: cross-lingual evaluations are sensitive to translation, locale and prompt language (Hofman et al, 2026), (Luo et al, 2026), (Kuang et al, 2026), (Li et al, 2026), (Agrawal et al, 2026). That literature measures the *agent* across languages; what we add is the complementary reading of the same manipulation as a property of the apparatus, localised to its components and checked for whether it moves success rate or only the reliability of the run.

2.6 Replication and artefact evaluation

Studies of the kind this paper reports are judged as much on whether a reader can rerun them as on what they found (Winter et al, 2022). Two conventions from that practice shape what we release. First, a replication package should carry the *frozen inputs*, not a recipe for regenerating them, because a recipe rerun months later reproduces a different system. Second, a package should distinguish what it can check from what it cannot: ours reports a per-source verification status and counts only sources



The oracle is the instrument that would have detected its own defect; it is therefore the one part the benchmark does not test.

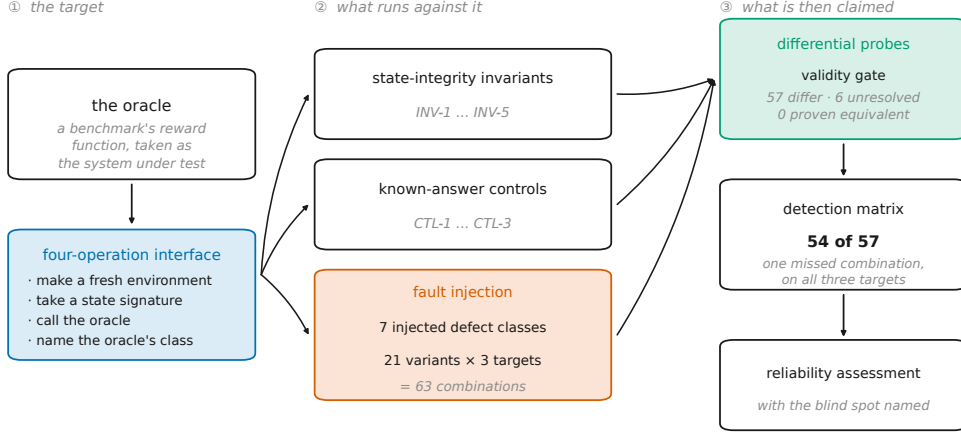
Fig. 1 The evaluation harness as the system under test. (a) An agent benchmark is a pipeline: a task suite, an input generator, the agent under test, an environment, and a reward function that acts as the *test oracle*. The oracle re-derives the expected state by replaying the reference solution into the same environment. In the implementation we audited it never restores what it wrote, so a harness that scores an episode *after* the session has ended reads the reference solution's state rather than the agent's (Section 4.1). (b) The same apparatus read two ways: the usual reading attributes the score to the model, whereas the reading this paper takes makes the harness the measuring instrument and the score a property of that instrument.

whose file was found *and* whose hash matched as verified, rather than treating an absent archive as a pass. Both choices are recorded in the deviation log rather than presented as the original plan.

3 Methodology

3.1 The evaluation harness as a system under test

The object of this study is not a model but the software that evaluates one, and the mapping to testing vocabulary is direct. The benchmark supplies a suite of *test cases*: tasks that specify a user's goal and, for each, a reference sequence of tool calls that satisfies it. The *test input* is generated at run time by a language model playing the user. The *system under test* is a tool-calling agent — a model plus a fixed control loop — and the *test oracle* is the benchmark's reward function, which decides pass or fail by comparing the terminal state of a database against the state the reference actions would have produced, and by requiring that certain output strings appear in what the agent said. The *assertions* are therefore two: a state comparison that is language-neutral, and a substring match on the final reply that is not.



The unit under test is the instrument, not the model: the suite asks whether the oracle would notice a defect in itself.

Fig. 2 The oracle-integrity auditing workflow. The unit under test is the oracle, taken as a program with a four-operation interface (**left**): make a fresh environment, take a state signature, call the oracle, and name the class the oracle lives in. Three families of checks run against that interface (**centre**): five state-integrity invariants, three known-answer controls, and a fault-injection battery of seven defect classes realised by 21 variants across three targets. A differential-probe validity gate decides which variant–target combinations count as separable *before* any detection number is computed (**right**); a combination the probes cannot separate is reported as unresolved and is never counted as equivalent. The suite detects 54 of the 57 separable combinations, and the one it misses on all three targets is named in Section 4.1 rather than summarised away.

Two properties of this arrangement matter for what follows. The oracle is a function of mutable state: to construct its expectation it must execute the reference actions, which means it must touch the same database it is grading. And the grading is a two-component conjunction, which lets us separate the state component from the textual one when the two disagree.

3.2 Oracle-integrity testing

The suite requires four operations of any object it audits: make a fresh environment for a given task, produce a comparable signature of the state being graded, call the oracle, and name the class the oracle lives in so that a defect can be injected. Nothing in the interface is specific to the benchmark under study.

On that interface we define eight checks in two families. The *invariants* state properties the oracle must satisfy whatever the task: **INV-1** a direct call must not change the graded state; **INV-2** two calls on the same environment must agree in verdict and leave the same state; **INV-3** the verdict must not depend on the order in which episodes are scored; **INV-4** if the oracle raises part-way through, the state must be restored and the exception must propagate; **INV-5** after a session-ending step the environment must still hold the state the agent left, and no actions beyond the terminating one may have been appended. The *controls* validate the direction of the verdict against a known answer: **CTL-1** an episode that accomplishes nothing, and one that says the required outputs without performing the required writes, must

not be graded as solved; **CTL-2** the reference solution must be graded as solved; **CTL-3** an accepted alternative spelling of a required output must be accepted.

To measure the suite’s own sensitivity and specificity we inject seven defect classes into the oracle, one at a time, each differing from a correct implementation in exactly one dimension: a state mutation that is never restored; a restore that happens only on the success path; a restore that forgets the recorded actions; a verdict that depends on an order through a module-level cache; an oracle that always passes; one that ignores the state component; and one that matches outputs without any normalisation. The unmodified oracle is the control for specificity.

3.3 Benchmark, task suite and configuration space

We use a tool-use benchmark of the “agent–user–database” form, in two of its domains: a retail domain, from which 30 of the test tasks were sampled for the pre-registered matrix, and an airline domain, whose 50 test tasks are used in full. The airline domain was added for this revision and required a Chinese version of its task instructions; that translation was produced by a model and then checked mechanically — identifiers, ISO dates, flight numbers, airport codes present in the English text, and every numeral must appear verbatim in the Chinese, and no Latin fragment absent from the English may appear — with the residual limitation that semantic equivalence rests on the mechanical check plus one reading pass, and not on a second independent translator.

Two factors are manipulated. The *user language* is the language in which the simulator writes. The *environment language* is the language of the three documents the agent reads: its policy wiki, its tool descriptions, and the strings its tools return on error. Database contents — names, addresses, order and product identifiers — are never translated, because they are data rather than prompt, and enumerated literals that a tool contract requires verbatim are kept in English inside the translated text, so that a localised arm cannot fail through argument validation instead of through language. The environment factor has a per-component form in which exactly one of the three documents is translated, and a length-padded form in which a semantically redundant appendix restores the Chinese prompt to the English length. Every arm is admitted by a gate that asserts, component by component, that the documents claimed to be translated are byte-identical to the project’s single set of translations and that those claimed not to be translated are the English originals by object identity.

3.4 Design and pre-registration

The pre-registered matrix varies the user language with the task, the agent, the simulator model and the repetition held fixed — 30 tasks $\times$ 2 languages $\times$ 6 simulators $\times$ 4 agents $\times$ 3 repetitions = 4,320 episodes — and is analysed from the archived episodes rather than regenerated. A second pre-registered factor crosses the user language with the environment language, 1,080 episodes, plus 1,080 in the length-padded environment. This revision adds three experiments, whose designs were frozen before the corresponding runs: a per-component ablation of the environment factor (3,240 episodes), a repeat of the user-language manipulation in a second domain

(1,200 episodes), and an independent adjudication of the collapse label (212 archived episodes).

Two arms were changed after their pilot data were seen, and we report this as a deviation rather than presenting the final design as the planned one. The second-domain experiment was to include a cross-vendor agent; its pilot collapsed in none of 120 episodes, so by the pre-registered stopping rule that arm was dropped, halving the experiment and leaving independent-model validation open. The adjudication was to draw 200 collapsed episodes; the archive contains 106, so it became a census. The pilot itself was a directional check with a pre-registered decision rule, not a significance test.

3.5 Measurement and statistics

The unit of randomisation we care about is the task, so the primary inference is a task-clustered interval: we aggregate the paired difference within each task, take the mean across tasks, and use the $t(G - 1)$ reference distribution, with G the number of tasks. Comparisons are paired within the same task, simulator, user language and repetition, so that a component arm is compared with the baseline on cells that differ in exactly the manipulated component. Exact and permutation tests are reported, where they appear, as cross-checks on an estimate and are never used on their own to declare an effect resolved. Every reported score is computed under the corrected grading described in Section 4.1; where an archived arm’s labels required reconstruction, we rebuilt the state component by replay and left the textual component as archived, since the defect affects only the former. An interval that contains zero is reported as no detected difference.

3.6 Replication package

The package accompanying this paper is assembled so that every number in it can be recomputed offline, on a machine with no network access, no API keys and no model client. It contains four parts.

- **The auditing suite and its fault catalogue.** The eight checks (Section 3.2), the seven injected oracle-defect classes, the positive and negative controls, and the frozen map from variant to defect class that the ablations aggregate over.
- **The frozen per-episode records.** The v1.0 main matrix, the component ablation and the second task domain, together with the archived labels and the corrected labels produced by the replay described in Section 4.1. Archived and corrected values are kept side by side rather than overwritten.
- **The pre-registration and the deviation logs.** The pre-registered design, and a numbered log recording every departure from it – including the two arms dropped on pilot evidence, the partition adjustment in the second benchmark, and the two defects found in our own validity gate. We record these rather than presenting the final grid as the planned one.
- **A single offline entry point.** One command runs the offline analyses in groups and prints an exit code per group: the engine-trace adjudication recomputation,

the audit ablations and baselines, the oracle-integrity regressions, and the source-binding check. The entry point reports which source archives it could verify and which were absent; it does *not* count an absent archive as verified.

Two things are deliberately *not* in the package. The second independently implemented benchmark used for the negative result in Section 6 ships its task data under a licence that forbids plaintext redistribution, so the package carries the adapter, the applicability findings and the version fingerprint, but not the data. And the model-served episode transcripts are published in aggregate and in the adjudication inputs, with the simulated user turns replaced by a placeholder; we do not claim that every raw provider response can be redistributed.

A container-free environment file pins the analysis dependencies; the model client is not among them. Nothing in the package calls a model, and the recomputation of every reported statistic is deterministic given the frozen records.

4 Results

4.1 RQ1 (primary): How reliable is the evaluation oracle?

The suite and its validation.

We implemented the eight checks of Section 3.2 against an interface of four operations — make a fresh environment, take a state signature, call the oracle, and name the class the oracle lives in — and ran them on three targets: the oracle of a retail-domain agent benchmark, the oracle of the same benchmark’s airline domain, and a synthetic domain written from scratch. The first two are *two task domains of one scoring implementation*, not two independent frameworks, and we do not claim more than that; the second domain contributes a different task suite, tool surface and database against the same oracle code. The synthetic domain exists to show that the suite is not shaped around one benchmark and to provide a ground-truth-controlled setting. Throughout, *three targets* means these three.

For each target we injected seven defect classes, one at a time, into the oracle. Table 1 gives the outcome.

The identity of the un-injected baseline matters, and we state it explicitly because the two sentences that follow would otherwise appear to contradict each other. The implementation this suite audits is the copy the package ships, and that copy carries the repair described below; the class labelled *as shipped* in Table 1 is therefore the *repaired* implementation, and it passes every check on every target. The class labelled *state mutation* is not a hypothetical: it is a faithful reproduction of the same function *before* the repair, with no restore statement of any kind. Against that reproduction, INV-5 fires on every task we sampled in both domains. A three-way check confirms the separation — the vendored source contains the snapshot-and-restore, the injected variant contains no restore at all, and INV-5 is silent on the first and alarming on the second in both domains — and it is released with the package. The suite would therefore have caught the defect had it been pointed at the benchmark as it stood before the repair; the claim is that it catches one of the seven classes, not that it fires on the code we ship.

What the class-level summary hides.

Table 1 summarises the injection experiment at the level of the seven defect classes, which is how the experiment was designed — and it is more generous than the result. Each class is realised by several concrete variants of the reward function; the battery is 21 variants, and running it against all three targets gives 63 variant–target combinations. On the *primary denominator* — the 57 combinations a differential probe showed to be behaviourally distinct from the shipped implementation, the remaining 6 being left unresolved by the probes and never counted as equivalent — the full suite detects **54 of 57**. On the *conservative denominator*, which also counts the 6 unresolved combinations, it detects 56 of 63. We report both and do not mix them: they answer different questions, and the larger number is not the stronger result.

The suite is therefore *not* free of blind spots, and the miss is the same combination on all three targets: an oracle that swallows an exception raised inside itself, so that INV-4 records the probed path as *not applicable* rather than as a violation. The mechanism is worth naming because it is not an accident of this implementation — a check whose precondition is destroyed by the defect it is meant to detect will record that defect as inapplicable. We report the miss rather than the class-level summary that conceals it.

Which checks are load-bearing.

The last column of Table 1 is computed rather than asserted. A check is *necessary* for a class when removing it leaves some detected combination of that class undetected by any remaining check; *none* means no single check is necessary, because two or more of them independently cover the class. Four of the eight checks are necessary for at least one class and four are not for any: INV-3 (9 unique detections across the battery), CTL-1 (15), INV-4 (6) and CTL-3 (5), against INV-1, INV-2, INV-5 and CTL-2 with zero. Redundancy is not a flaw in a test suite — an independent check that agrees is how a suite earns trust — but it does mean the eight checks are not eight independent capabilities, and the detector ablation in the replication package is computed on that basis. The two families remain non-interchangeable: an oracle that is state-clean but wrong is invisible to the invariants, and one that is state-dirty but accidentally right is invisible to the controls.

What the suite found on the benchmark.

The benchmark’s reward function resets the environment database in order to replay the reference solution and construct a ground-truth hash, and it never restores the database afterwards. The value the function returns is correct, and the benchmark’s own agent consumes that value without reading the environment again, so the benchmark’s own runner is unaffected. What turns the side effect into a wrong label is a harness that scores the episode separately, after the session has ended: the second read finds the reference solution’s state rather than the agent’s. Our harness did exactly that.

Invariant INV-5, terminal-state stability, fires on that reproduction for this reason, and it fires without any downstream code being involved: after a session-ending step the environment no longer holds the state the agent left. We keep the two phases

Table 1 Defect class $\times$ detector, computed from the released fault matrix over all three targets. **Fires** lists every check that alarmed on at least one variant of that class; **necessary** is the check whose removal leaves some detected variant of that class undetected by any remaining check. Variants within a class do not all produce the same alarm pattern, which is why the table reports a union rather than a single pattern; the class *saturated oracle* contains one variant, on each target, that no check alarms on at all (Section 4.1). Every variant is author-constructed, so these figures are a conditional property of this battery, not a detection rate for unknown defects.

Defect class	variants	checks that fire	necessary
state mutation	9	INV-1, INV-2, INV-4, INV-5, CTL-1, CTL-2	CTL-1
exception leak	9	INV-1, INV-2, INV-4, INV-5, CTL-2, CTL-3	INV-4
actions not restored	9	INV-1, INV-2, INV-4, INV-5	<i>none</i>
order dependence	9	INV-3	INV-3
saturated oracle	9	CTL-1	CTL-1
database-blind oracle	7	CTL-1	CTL-1
strict output match	5	CTL-3	CTL-3

apart: the *discovery* of the defect was made on the unpatched implementation during the earlier study, whose episodes are the archived ones analysed below; the *validation* of the detector is the injection experiment above, in which the unpatched logic is reproduced deliberately. INV-5 is silent on the repaired copy we ship, and that silence is the specificity result, not a contradiction of the discovery. The counter-example is minimal. We construct an episode in which the agent performs no write operation at all and simply says goodbye; on the archived code path this episode is graded as a database pass, and after the repair it is not.

The correction.

Because the benchmark exposes no deterministic seed and the provider does not offer immutable model versions, the 4,320 archived episodes cannot be regenerated; they can, however, be *replayed*. We re-execute each archived tool call against a clean database and compare the resulting terminal hash with the reference hash. Two checks establish that the replay is faithful: all 137 episodes that ended by hitting the step limit — which never pass through the defective path, and whose labels were therefore already correct — reproduce their archived labels exactly, and all 33,402 tool observations reproduce the archived strings character for character, with no mismatch.

Under the corrected labels the archived 4,194 database passes become 2,883, and overall success falls from 94.3% to 65.5%. Every number reported below is computed under the correction.

4.2 RQ2 (secondary): How do harness configurations influence observed evaluation outcomes?

The agent’s own world is written in three documents: the policy it reads, the descriptions of its tools, and the strings its tools return on error. Manipulating all three at once tells us that the world matters but not which part of it; we therefore ran each component alone. Three arms localise the policy, the tool descriptions, or the error strings into Chinese and hold the other two in English, over 30 tasks $\times$ 6 simulators

Table 2 Component ablation of the environment localisation (retail, 1,080 paired cells per arm). Δ is the paired difference against the fully English baseline, with a task-clustered 95% interval ($G=30$, $t(29)$). Success differences are computed under the corrected scorer.

Arm	Collapse	Δ collapse	Δ success
English baseline	9.81% (106/1080)	—	—
Tool descriptions Chinese	0.65% (7/1080)	−9.17 [−11.61, −6.73]	+9.63 [+5.09, +14.17]
Chinese environment (all three)	1.76% (19/1080)	−8.06 [−10.57, −5.54]	+5.56 [+0.45, +10.67]
Chinese + length padding	1.30% (14/1079)	−8.52 [−10.77, −6.28]	+5.80 [+0.56, +11.04]
Policy Chinese	2.59% (28/1080)	−7.22 [−9.46, −4.98]	+5.09 [+0.80, +9.38]
Error strings Chinese	4.54% (49/1080)	−5.28 [−8.03, −2.52]	+5.19 [+1.31, +9.06]

$\times 2$ user languages $\times 3$ repetitions = 3,240 episodes, with no failed cell. Two archived arms — the fully English baseline and the fully Chinese environment — are reused rather than re-run, and every comparison is paired within the same (task, simulator, user language, repetition) cell and clustered by task.

Table 2 gives the result. All four Chinese arms have intervals that exclude zero, and the ordering is stable: localising the tool descriptions has the largest effect, the policy the next, and the error strings the smallest, with every pairwise difference also excluding zero.

Two claims follow, and both are narrower than the one the headline of an earlier draft of this paper would have supported.

First, **all three components carry a protective effect, and the effects do not add**. The weakest single arm still moves collapse by −5.28 pp. Each component therefore changes the measured collapse rate in the tested configuration when it is localised on its own; but these single-component contrasts do not establish that any of the three is *necessary*, and we do not claim that. But the strongest single arm (tool descriptions, −9.17 pp) has a larger point estimate than the arm that changes all three documents (−8.06 pp), so it is *not* the case that no single component reaches the full effect — the full manipulation fails to beat its own largest component. Comparing two intervals side by side would not settle this, since two intervals that each exclude zero need not differ from one another; we therefore test the contrast directly, pairing the two arms cell by cell. Localising only the tool descriptions is better than localising all three by 1.11 pp, with an interval of [0.36, 1.86] that excludes zero. We do not read this contrast as a mechanism ranking, for two reasons given below and in Section 6: it is small, and it is not a simultaneously run comparison — the three single-component arms ran in one batch and the all-three arm comes from an earlier batch of the same project, so the contrast is paired on task, simulator, user language and repetition but is not a within-session randomisation. Quantitatively: a simulation on this design’s own variance puts a contrast of 1.11 pp at **17%** power, against 85% at 4 pp and 100% at 9 pp. On the evidence we have, this contrast is *undecidable at this design’s resolution* rather than resolved in favour of either arm, and the interval that excludes zero should not be read as settling it. Localising only the error strings is worse than localising all three by 2.78 pp [1.21, 4.34], and the policy alone does not differ from the full manipulation (+0.83 pp [−0.36, +2.03]). Adding the policy wiki and the error strings to a localised tool description therefore takes back part of the gain rather than adding

to it. We describe this as a *non-additive observed pattern* rather than as sub-additivity, because with only the single-component arms and the all-three arm we cannot identify the interaction terms of higher order; what the design supports is the observation that the components do not combine as independent contributions, and that the ordering has a largest and a smallest term rather than a cause. The component contrasts are exploratory: the pre-registration set no hypothesis on differences between components, and we report the intervals without a multiplicity correction, so they should not be read as a definite ranking of the three documents.

Second, the error strings are the *weakest* component. We had pre-registered the opposite: the mechanism we had described earlier located the failure at the tool-error boundary — the identity check that returns `user not found` — and we therefore expected the error vocabulary to carry the effect. It carries the least of the three. The earlier mechanism account and this result do not agree, and we report them as disagreeing rather than reconstructing a story that fits both.

Length.

Localising a document also changes its length, and for Chinese it shortens it: measured in characters, the Chinese policy wiki is 36% of the English length, the tool descriptions 66%, and the error strings 30%. A length-matched fourth environment, padded back to the English prompt length, moves collapse by -0.46 pp and success by $+0.24$ pp relative to the unpadded Chinese environment — one order of magnitude below any component effect. We detect no length effect at this resolution, and accordingly we do not treat length as the explanation of the component effects; the interval for the padded arm is reported with the other arms and crosses zero. Length is not, however, fully excluded for the single-component arms: the padding controls the length of the environment as a whole, and we did not build a length-matched counterpart of each individual document. A per-component length confound of the observed size would have to be an order of magnitude larger than the whole-environment length effect to overturn the ranking, which we regard as implausible but do not rule out.

A second domain reverses the sign.

We repeated the user-language manipulation on the benchmark’s airline domain — a different task suite, a different tool surface, and a substantially harder set of tasks — over $50 \text{ tasks} \times 6 \text{ simulators} \times 2 \text{ user languages} \times 2 \text{ repetitions} = 1,200$ episodes, with no failed cell. The domain is where we had hoped to give the stronger agents a non-zero base rate: it does not do that (the cross-vendor agent we had planned to use collapsed in none of 120 pilot episodes, so that arm was dropped and independent-model validation remains open, see Section 6).

The agent whose base rate is non-zero does collapse in the second domain, at 9.67% (English simulator) and 4.33% (Chinese simulator). The direction, however, is the opposite of the one we measured before:

Domain		English simulator	Chinese simulator
retail (archived)	collapse	6.48%	13.15%
airline (this study)	collapse	9.67%	4.33%

The retail figures are per user-language arm of 540 episodes for that agent — 1,080 is the two arms combined — and the airline arm is 600 paired cells per language. The airline difference is -5.33 pp with a task-clustered interval of $[-8.15, -2.51]$, which excludes zero; the retail difference is $+6.67$ pp. Reporting two intervals separately does not establish that the two domains differ, so we also test the contrast directly: the difference between the two effects is -12.00 pp with an interval of $[-16.33, -7.67]$ (Welch–Satterthwaite, $df = 65.76$; exact t quantile), which excludes zero. The signs are opposite, each is resolved, and so is the difference between them. We pre-registered the two ways this could be read, and this is the second: the effect does not generalise, so we report it as a boundary and restrict the earlier finding to the configuration in which it was measured — a language effect *under the retail configuration*. On the success side the airline difference is -2.00 pp $[-7.05, +3.05]$, which excludes nothing and supports no direction.

This is a negative result and we report it as one, and we keep it to what two domains can support. The sign is not stable across the domains we ran, so no direction can be attributed to the simulator’s language; the effect is a property of the configuration in which it is measured. That is weaker than a mechanism claim, and we do not make one: two domains, each changing its task set, tool surface and base rate together, cannot separate a configuration property from an interaction between language and domain. What the reversal rules out is the reading the earlier literature invites — that Chinese makes an agent worse or better — and that is the claim we retire.

4.3 RQ3 (secondary): How sensitive are the conclusions to execution factors, and where do they stop holding?

RQ1 and RQ2 report what the audit found. RQ3 reports where each of those answers stops holding: the *implementation* boundary, at which the same auditing method failed to transfer to a second, independently implemented framework; and the *instrument* boundary, at which the criterion this paper relies on turns out to be an operational definition corroborated by a language model rather than a human ground truth. We report both as results rather than as omissions. The task domain is itself part of the configuration, and we treat its effect under RQ2 for that reason.

The implementation boundary.

We applied the same oracle-integrity suite to a second, independently implemented agent framework. The attempt did not yield a validation. The reference terminal state that the positive control needs cannot be reconstructed by an external harness from that framework’s recorded API-call log, so no sensitivity or specificity could be estimated on its held-out task set and we report none. The fixture the framework does expose cannot serve as a positive control either: of its 21 held-out tasks, the three that look usable are false positives produced by the same state-reconstruction defect the control exists to detect. The constraint is contractual rather than incidental — the framework materialises a reference episode only inside its own run loop — and we state it as the precondition a framework must meet before this class of auditing can be ported to it. Section 6 records it as the external-validity boundary it is.

An independent language-model adjudication reproduces the collapse label.

The collapse criterion is a mechanical operational definition, and a reader is entitled to ask whether an independent judge would call the same runs degenerate. We put every collapsed episode in the archive (106) together with a matched non-collapsed control, masked the language arm and the criterion’s own intermediate quantities by replacing each user turn with a placeholder, and had a judge from a different model family decide, blind to the hypothesis, whether the agent stops making progress. Of 212 items the judge returned a parseable verdict on 210 (0.9% attrition). Agreement with the mechanical criterion is 98.1% (Wilson 95% [95.2, 99.3]), Cohen’s κ is 0.962, sensitivity is 98.1% and specificity 98.1%.

The quantity that matters for this paper is not the agreement but its *independence of the language arm*: 98.6% in the English arm (one discordant item) and 97.9% in the Chinese arm (three discordant items). A criterion that was tuned, however inadvertently, to fire in one language would show up here. It does not. The figures are agreement with the mechanical criterion and not accuracy against a human reference: the adjudicator is a language model, and the sample is the archive’s positives plus matched negatives rather than a draw from a deployment distribution, so the sensitivity and specificity we report are properties of that selected sample and do not transfer to a population with a different base rate. Three limits therefore bound this result. Agreement is at the ceiling, so the test has little power to detect a moderate arm-dependent bias; and the residual leakage is real — to judge whether the agent stops making progress the judge must read what the agent said, and in the Chinese arm those words are Chinese — so we claim that the criterion’s agreement does not depend on the arm, not that the judge is blind to language. The judge is also a language model, not a person; this study has no annotator budget, so the criterion is reported as an operational definition with independent corroboration rather than as a natural kind.

5 Discussion

An evaluation oracle is software, and it needs the tests other software gets.

The defect we report is not exotic. A function that mutates the state it is grading, and restores it on the success path but not on the failure path, is the ordinary shape of a resource-handling bug; what makes it consequential here is that the function is the grader, and that its caller had no reason to suspect it. Two of the checks in our suite would have caught it immediately and cost nothing to run. The repeat-scoring check — score the same finished episode twice and require the same verdict — fails on the first call after the first, because the first call has already moved the environment. The terminal-state check — after a session-ending step, assert that the environment still holds the state the agent left — fails without any downstream code being involved. Neither requires a reference implementation, a model call, or a human. We suggest that both belong in the continuous integration of any harness that grades an agent episode, next to the regression test that pins the fix.

What a cross-lingual score table does not say.

A score table reports a measurement. It does not report the instrument’s failure rate, and our results are a case in which the instrument’s failure rate and the measurement move independently: in the retail configuration the aggregate score difference is unresolved while the rate at which one agent’s trajectory degenerates roughly doubles, and in the airline configuration the same manipulation moves the degeneration rate in the opposite direction. A practitioner comparing two language configurations of the same evaluation therefore cannot read the comparison as a statement about the model, and cannot even assume the direction of the instrument’s bias is stable across task suites. Reporting an instrument-failure rate alongside the score, and checking it in more than one task suite before acting on it, are the two actions that follow.

Localisation is a configuration change, and it should be tested as one.

The practical reading of the component ablation is not that one document matters and the others do not; it is that changing the localisation of an agent’s world changes the reliability of the measurement, and that the change is spread over the parts of the world the agent reads. A team that localises the user-facing side of an agent while leaving its policy, its tool descriptions and its error vocabulary in English is running a configuration whose measurement properties differ from both the fully English and the fully localised ones. Testing that configuration — rather than assuming it is an intermediate point between the two — is what the ablation suggests.

Replay as the audit primitive.

The correction we report is not a re-run: the provider offers no deterministic seed and no immutable model version, so the episodes cannot be regenerated. What can be regenerated is the *effect* of the agent’s recorded actions on the state the oracle grades. Replaying archived tool calls against a clean database reconstructs the terminal state exactly, and it is what turned an inflated label into a measurable one. The precondition is narrow and we state it as a precondition rather than as a prediction: the replay needs a benchmark that records the agent’s tool calls in a form an external harness can re-execute, against a database the harness can reset. We evaluated it on one such benchmark, and Section 6 reports the independent implementation on which the same precondition did not hold. Within that precondition it is cheap: the replay is deterministic, needs no model, and can be cached.

Reporting a negative result, and reporting a disagreement.

Two of this paper’s findings are negative: a pre-registered mechanism hypothesis was refuted by the ablation, and a previously reported effect changed sign in a second task domain. We report both against the interest of the narrative, and we report the first as an explicit disagreement between an earlier mechanism account and the new data rather than reconstructing an account that fits both. For a paper whose subject is the trustworthiness of measurements, this is the only consistent position.

6 Threats to Validity

6.1 Construct validity

What RQ2 and RQ3 are not.

We state the boundary of the two secondary questions explicitly, because it is the reading a reader is most likely to reach for. RQ2 and RQ3 are *not* measurements of intrinsic language-model capability or behaviour. The models are the materials of the study, not its object. What those questions characterise is the measurement process on an audited pipeline: how design choices in the harness expose, amplify or obscure variability that is present while a benchmark runs, and where the resulting conclusions stop holding. The consequence for the evidence is stated in the Introduction and carried through the paper: RQ1 is direct evidence about the evaluation system, and RQ2 and RQ3 are reported as observations and boundaries at correspondingly weaker strength. A reader who wants a claim about how a model behaves under localisation will not find one here, and we do not make one.

Collapse is an operational definition: a maximal run of at least five consecutive reply steps each spending at most two completion tokens, with no tool call. It is mechanical and reproducible, and it is not a natural kind. The independent adjudication in Section 4.3 establishes that the definition tracks what a reader would call a stalled trajectory, and that its agreement does not depend on the language arm, but the judge is a language model rather than a person and agreement is at the ceiling. The criterion is also defined on the agent’s token counts, which are not a language-neutral unit of text; earlier work in this line showed that the gap survives character-count, repetition and no-length-threshold alternatives, and we rely on that robustness rather than claiming the token threshold is comparable across languages. What we measure is therefore a property of the evaluated agent–simulator *configuration* — how often, under this configuration, a trajectory degenerates — and not a property of the model alone.

6.2 Internal validity

The provider exposes no effective deterministic seed: at temperature zero the same request gives different trajectories, so the three “repetitions” per cell are a repetition index and not a controlled randomisation, and a single run is not evidence. All comparisons are therefore paired within a cell and reported with task-clustered intervals. A further internal-validity cost is that the component ablation and the second domain ran on a platform that rebuilds the execution environment periodically; the ablation batch consequently spans one such boundary. The elapsed gap was under an hour, and a drift probe run over intervals an order of magnitude longer than that gap found no evidence of version drift, but we cannot exclude it.

A larger instance of the same threat attaches to the component contrasts. The three single-component arms were run in a single batch, but the arm that localises all three documents and the fully English baseline come from earlier batches of this project; the gap between the run periods is at least a day, and the provider exposes no immutable model version, so a provider-side change between the periods is not

excluded. Pairing on task, simulator, user language and repetition removes task and configuration variance, but it is not a within-session randomisation and cannot remove drift common to a period. We state the consequence precisely. The *existence* of a protective effect for each component is supported; the *ordering* of the components should be read as an observation in this configuration rather than as an established mechanism, and the 1.11 pp contrast between localising the tool descriptions alone and localising all three is the most exposed of the contrasts we report. A same-period replication with the arm order reversed would settle it, and we did not run one.

Two arms of the second experiment were dropped on the evidence of their pilot (Section 4.2), which is a design decision made after seeing data, and we record it as such rather than presenting the final grid as the planned one.

6.3 External validity

The auditing method has a portability boundary that we measured rather than assumed. We applied the same suite to a second, independently implemented agent benchmark. The attempt is a *negative* result: the reference terminal state needed by the positive control could not be reconstructed from the framework’s recorded API-call log, so no detection rate could be estimated on its held-out task set, and we report none. The constraint is contractual rather than incidental — the framework materialises a reference episode only inside its own run loop — and we state it as the precondition a framework must meet before this class of auditing can be ported to it. This bounds Contribution 1 to the implementations we actually evaluated.

All the remaining results come from one agent benchmark with two task domains, four agents and six simulator models, all served by one provider. The trajectory-level effect is carried by one agent; the others have a zero base rate in both arms and cannot show it, so the mechanism’s generality across models is inferred from its mechanics rather than observed. The second domain was intended to relax this and did not: the cross-vendor agent collapsed in none of its pilot episodes, so the arm was dropped and *independent-model validation remains open*. The cross-lingual setting is a simulator speaking Chinese against an English tool environment; other language pairs are untested, and the simulated users are language models rather than human speakers, which leaves open how much of the effect belongs to the simulator’s language behaviour and how much to the agent’s. Inference is conditional on the sampled tasks: the retail results are conditional on the 30 tasks drawn from the benchmark’s test suite, and the airline results on all 50 of that domain’s tasks.

6.4 Conclusion validity

The primary inference is a task-clustered interval, and the reference distribution is $t(G - 1)$ with G the number of tasks, which treats tasks as the sampling unit. This is the right unit for claims about a task suite and the wrong unit for claims about episodes; the intervals we report are conditional on the tasks sampled and do not absorb the sampling step. Exact tests are reported only as a cross-check and are not used to declare an effect resolved. Several of our secondary analyses — permutation, sign-flip and task-clustered bootstrap variants — do not share a single null hypothesis

and therefore do not constitute independent lines of evidence; they are reported as sensitivity analyses of one estimate. Finally, an interval that contains zero is reported as no detected difference and not as evidence of no difference.

7 Conclusion

An agent evaluation harness is a piece of software, and the score it produces is an instrument reading. We audited one such instrument and report two findings. Its test oracle mutated the state it was grading, which inflated the recorded success rate from 65.5% to 94.3% in our runs and would have done so silently; the mechanism is a missing restore, the detection is a repeat-scoring or terminal-state check, and the correction is a replay of the recorded actions rather than a re-run. And the language of the apparatus is a configuration parameter of the measurement rather than a property of the model: localising the agent’s own world changes the rate at which trajectories degenerate, each of the three documents the agent reads carries a protective effect, no single one of them accounts for the other two, and the effects do not add. The sign of that effect does not survive a change of task domain. A score table that reports neither the instrument’s failure rate nor the configuration it was taken under is reporting less than it appears to.

Declarations.

- **Funding.** No funding was received for conducting this study.
- **Competing interests.** The author declares no competing interests.
- **Ethics approval.** This study involves no human participants and no animal subjects. The “users” in all experiments are language-model simulations; no human subjects were recruited, tested, or recorded.
- **Data availability.** The software artifact and the frozen evaluation dataset are archived separately on Zenodo. The software artifact is at <https://doi.org/10.5281/zenodo.23290600> (v1.0.0). It contains the oracle-integrity suite, the fault-injection battery, the analyses that produce the reported tables and data-derived figures in this paper, the version-binding manifests, and the small frozen inputs those analyses read.

Behind those analyses are four frozen archives: the v1.0 main experiment (4,320 episodes) with both the archived and the replay-corrected labels; the S2 release holding the component ablation (3,240 cells), the cross-domain experiment (1,200 cells) and the blind-adjudication inputs (212 items); the enhancement round holding the held-out variant battery, the detector ablations, the power analysis and the cross-framework portability attempt; and an anonymised copy of the first. They total 232 MB and are archived separately at <https://doi.org/10.5281/zenodo.23290703>, together with the data terms they are released under and a checksum file. Each number in this paper is bound to exactly one of those archive revisions, and the mapping from archive to experiment to section, with SHA-256 checksums, is recorded in the package’s release manifest.

Two things are deliberately not distributed. The second, independently implemented agent framework used for the portability result ships its task data under a licence that permits redistribution only in encrypted form, so the package carries

the adapter, the applicability findings and the version fingerprint, and this paper reports no detection rate for it. The model-served episode records are distributed as frozen records; they contain outputs of third-party model services and are not offered under an unconditional open licence.

The code authored for this study is released under the **MIT** licence. The frozen records are *not* covered by that licence and carry their own terms, stated in the package: the derived results and the per-episode records may be used and redistributed for research with attribution, while large-scale redistribution or commercial use of the episode records should be requested first, because they contain outputs of third-party model services.

Note on byte-reproducibility: the analyses are deterministic and recomputable from the frozen records, but the *PDF* is not byte-reproducible, because the $\text{\LaTeX}$ engine writes a build timestamp and a document identifier into every file it produces. The published PDF should therefore be bound to the released source by a hash of the source, not of the PDF.

- **Code availability.** The analysis and experiment code is contained in the archives above and is also available at <https://github.com/JunjieChen0/SimGlot> (tag v1.0.0). Three entry points drive every offline analysis: one for the S2 side, one for the enhancement side, and one for the v1.0 analyses. The oracle-integrity suite runs from a single command and needs no network, no credentials and no model; it audits the benchmark’s own oracle and prints the defect-class $\times$ detector matrix reported in Table 1. The end-to-end analysis is reproducible from the archived episodes by one further command that neither reads an API key nor opens a network connection, and that refuses to run if a key is present in the environment, so that a claimed offline reproduction cannot silently become an online one. The suite’s interface is four operations wide, so it can be pointed at another harness’s oracle without changing the suite itself.
- **Use of large language models.** Language models are the experimental material of this study: they act as the tool-calling agents under test, as the user simulators that generate the test inputs, as the translator that produced the Chinese task instructions (with the mechanical fidelity check described in Section 3.3), and as the independent adjudicator in Section 4.3. The models are specified by name and version throughout, and the prompts given to the adjudicator and to the translator are released with the package. Language models were also used to assist in drafting the manuscript and the analysis code; the author reviewed and is responsible for all content, analyses and conclusions.

References

- Agrawal K, Feng Y, Leung MF (2026) Translating the translator: Decomposing the cost of english-forced inter-agent communication. URL <https://arxiv.org/abs/2609.15079>
- 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 (2026) τ^2 -bench: Evaluating conversational agents in a dual-control environment. In: International Conference on Machine Learning (ICML), URL <https://arxiv.org/abs/2506.07982>
- Bellibatlu RR, Wang Z, Zhang W (2026) Do agent benchmarks do what they say? an executable-contract audit of tool-using agent environments. URL <https://arxiv.org/abs/2609.37315>
- Bjarnason BH, Silva A, Monperrus M (2026) On randomness in agentic evals. URL <https://arxiv.org/abs/2602.07150>
- Chen TY, Kuo FC, Liu H, et al (2018) Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys 51(1):4:1–4:27. <https://doi.org/10.1145/3143561>
- Dou Y, Galley M, Peng B, et al (2025) Simulatorarena: Are user simulators reliable proxies for multi-turn evaluation of ai assistants? In: Proceedings of EMNLP, URL <https://aclanthology.org/2025.emnlp-main.1786/>, code: <https://github.com/microsoft/SimulatorArena>
- Haq I, Saldías B (2026) Dialect vs demographics: Quantifying llm bias from implicit linguistic signals vs. explicit user profiles. In: ACM Conference on Fairness, Accountability, and Transparency (FAccT), URL <https://arxiv.org/abs/2604.21152>
- Hofman O, Brokman J, Rachmil O, et al (2026) Maps: A multilingual benchmark for agent performance and security. In: Findings of EACL, URL <https://aclanthology.org/2026.findings-eacl.42/>
- Hou X, Zhao Y, Liu Y, et al (2024) Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology 33(8):220:1–220:79. <https://doi.org/10.1145/3695988>
- Jain A, Sandhu A (2026) Userproxybench: Evaluating llm user simulators for agent benchmarks and training. URL <https://arxiv.org/abs/2609.38043>
- 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>
- Kehkashan T, Abdullah M, Al-Shamayleh AS, et al (2026) From benchmarks to deployment: a comprehensive review of agentic ai evaluation. Artificial Intelligence Review <https://doi.org/10.1007/s10462-026-11571-0>
- Khanal A, Tao Y, Zhou J (2026) Beyond pass@1: A reliability science framework for long-horizon llm agents. URL <https://arxiv.org/abs/2603.29231>

- Khosravi H, Huo X (2026) Rank confidence sequences: Anytime-valid leaderboards. URL <https://arxiv.org/abs/2609.32211>
- von Kistowski J, Arnold JA, Huppler K, et al (2015) How to build a benchmark. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE), pp 333–336, <https://doi.org/10.1145/2668930.2688819>
- Kitchenham BA, Pfleeger SL, Pickard LM, et al (2002) Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering 28(8):721–734. <https://doi.org/10.1109/TSE.2002.1027796>
- Kuang P, Fan Y, Li J, et al (2026) Babelarena: A large-scale multilingual benchmark for llm agents. URL <https://academy.dair.ai/papers/babelarena-a-large-scale-multilingual-benchmark-for-llm-agents-2609.23490>
- Lee Y, Jin H, Kim TS, et al (2026) Knowsim: Evaluating information calibration in llm assistants with user simulators that learn. URL <https://arxiv.org/abs/2608.17150>
- Li H, Liu Y, Zhang Z, et al (2026) Polyworkbench: Benchmarking llm agents for cross-lingual long-horizon workflows. URL <https://arxiv.org/abs/2607.06008>
- Liu Y, He Y (2026) Disclosure-gated user simulation for companion-agent evaluation. URL <https://arxiv.org/abs/2609.00982>
- Luo Q, Hariri F, Eloussi L, et al (2014) An empirical analysis of flaky tests. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp 643–653, <https://doi.org/10.1145/2635868.2635920>
- Luo Z, Kutralingam TP, Okoani ON, et al (2026) Lost in execution: On the multilingual robustness of tool calling in large language models. In: Proceedings of ACL, URL <https://aclanthology.org/2026.acl-long.2039/>
- Ma C, Wanyan X, Hettiachchi D, et al (2026) Verifiable user simulation for search and recommendation systems. <https://doi.org/10.1145/3805712.3808645>
- Mahmood A, Abdaljalil S, Kurban H (2026) Multilingual prompt localization for agent-as-a-judge: Language and backbone sensitivity in requirement-level evaluation. URL <https://arxiv.org/abs/2604.04532>
- Ni B, Wang Y, Wang L, et al (2026) A survey on llm-based conversational user simulation. In: Proceedings of EACL, URL <https://aclanthology.org/2026.eacl-long.200/>
- Recht B, Roelofs R, Schmidt L, et al (2019) Do ImageNet classifiers generalize to ImageNet? In: Proceedings of the 36th International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research, vol 97. PMLR, pp 5389–5400

- Sekulić I, Terragni S, Guimarães V, et al (2024) Reliable llm-based user simulator for task-oriented dialogue systems. URL <https://arxiv.org/abs/2402.13374>
- Seshadri P, Cahyawijaya S, Odumakinde A, et al (2026) Lost in simulation: Llm-simulated users are unreliable proxies for human users in agentic evaluations. In: Proceedings of ACL, URL <https://aclanthology.org/2026.acl-long.2192/>
- Sierra Research (2024) τ -bench: A benchmark for tool-agent-user interaction in real-world domains (code repository). <https://github.com/sierra-research/tau-bench>
- Taday E, Cima L, Fagni T, et al (2026) Assessing the reliability of persona-conditioned llms as synthetic survey respondents. In: Companion Proceedings of the ACM Web Conference, URL <https://arxiv.org/abs/2602.18462>
- Tonneau M, Sehgal NKR, Malhotra N, et al (2026) Different demographic cues yield inconsistent conclusions about llm personalization and bias. URL <https://arxiv.org/abs/2601.18486>
- UserSim Organizers (2026) Usersim workshop: Grounded user simulation for model evaluation and training (neurips 2026 workshop). <https://usersim-workshop.github.io/>, 2026-12-12, Paris
- Winter S, Timperley CS, Hermann B, et al (2022) A retrospective study of one decade of artifact evaluations. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp 145–156, <https://doi.org/10.1145/3540250.3549172>
- Yehudai A, Eden L, Li A, et al (2026) A survey on evaluation of llm-based agents. In: Findings of ACL, URL <https://aclanthology.org/2026.findings-acl.1330/>
- Yin S (2026) Does the judge prefer english? evaluating language-switching invariance in llm-as-a-judge. URL <https://arxiv.org/pdf/2606.14278>
- Zhang JM, Harman M, Ma L, et al (2022) Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering 48:1–36. <https://doi.org/10.1109/TSE.2019.2962027>
- Zheng Q, Yang Y (2026) Are near-tied llm rankings robust to family-dif-guided benchmark recomposition? URL <https://arxiv.org/abs/2609.00482>
- Zhou X, Zhu H, Mathur L, et al (2023) Sotopia: Interactive evaluation for social intelligence in language agents. URL <https://arxiv.org/abs/2310.11667>
- Zhou X, Sun W, Ma Q, et al (2026) Mind the sim2real gap in user simulation for agentic tasks. URL <https://arxiv.org/abs/2603.11245>