Auditing by Re-Solving: LLM Benchmark Auditors Trust Their Own Answer Over the Reference

Anonymous ACL submission

Abstract

Benchmark items have several parts — an instruction, source records, a reference answer, and evaluator code — and any of them can be wrong. A natural response is to ask an LLM to audit the item; but this creates a circular measurement problem, since evaluating the auditor seems to require another auditor. We sidestep this by constructing audit tasks whose correct verdict is known by design: 200 web-agent-style benchmark items (each a question over ∼800 structured records), rendered four ways (clean, or with exactly one defect injected into the instruction, reference, or evaluator), so an audit can be scored mechanically against where we put the defect. Across 2,400 audits from three production models, we find that the auditor's check of the reference answer is only as reliable as its own ability to compute that answer itself. When the benchmark requires tallying hundreds of records, detection of a wrong reference falls from 68% to 9% and false positives on clean items rise from 44% to 88%; detection of a buggy evaluator, found by reading code rather than recomputing, stays at 80%, so the failure is not general difficulty. Reasoning traces and an answer-supplied probe converge on the mechanism: when checking a reference answer, the auditor often re-solves the task and trusts its own answer over the reference.

1 Introduction

Benchmarks have flaws: instructions are underspecified, reference answers are wrong, evaluators accept incorrect outputs. The flaws are hard to catch — SWE-bench was rebuilt as SWE-bench Verified under multiple rounds of expert review (Jimenez et al., 2024; OpenAI Preparedness Team, 2024), and OpenAI later stopped evaluating on it because residual underspecified tasks, broken evaluators, and hidden test leakage continued to corrupt the signal (OpenAI, 2026). Audits across nine widely-used benchmarks flag problematic items at rates that justify expert review (Truong et al., 2025); in test-based coding benchmarks specifically, strengthened test suites show that a large fraction of "solved" SWE-bench items are semantically incorrect, passing only because the evaluator is too weak to expose them (Yu et al., 2025; Wang et al., 2026; Yu et al., 2026). If multi-round human verification still misses defects, the operational response is to deploy frontier LLMs as automated auditors (Tu et al., 2026). This raises a measurement question: an LLM auditor returns an evidence-citing verdict — but is the verdict correct, and when it is wrong, *how*?

We sidestep this by fixing the right audit answer by construction. An audit, in our setup, is a single closed-context call: the auditor sees the artifact (instruction, source records, reference, evaluator) with stable line IDs in one prompt and returns a defect category and a list of cited line IDs; nothing is iterated and no tool is used. We build a corpus of 200 web-agent-style scenario families (each a question over ∼800 structured records), rendered as four matched variants (clean, plus a single defect placed in the instruction, the reference answer, or the evaluator), with the source records held fixed across variants. Because we inject the defect ourselves, each item has a known defect category and known evidence lines, and we score an audit as correct only when it names both (Section 2) — without any judge. (Citation *existence* is trivial here since cited lines are in the prompt with stable IDs; reaching the construction-known evidence is not.)

Our central finding is that the auditor's check of the reference answer is only as reliable as its own ability to do the underlying task itself. Tasks split into two groups: *count* tasks, where the answer is a single integer aggregated over hundreds of qualifying records, and *non-count* tasks, where the answer is one or a few specific records picked out of the source. On non-count tasks — where the auditor can reliably solve the task — it correctly

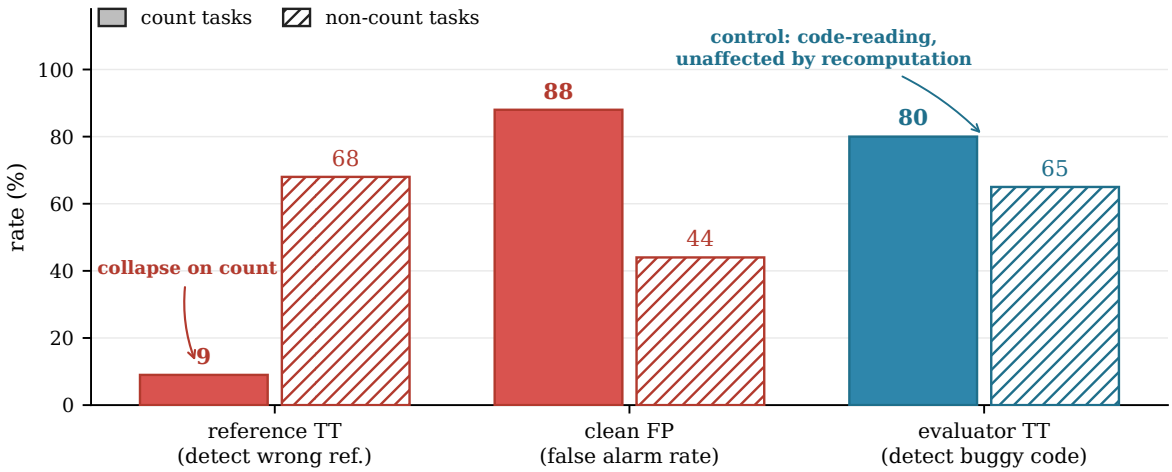

Figure 1: **Reference checking fails when it requires recomputation.** On count tasks, where the answer is a tally over hundreds of records, auditors rarely detect wrong reference answers and often flag clean items as defective. By contrast, detection of buggy evaluators remains high, because those defects can be found by reading evaluator code rather than recomputing the answer. Bars show parse-valid rates pooled across three production models.

detects a wrong reference 68% of the time and false-alarms on clean items at 44%. On count tasks — where the auditor cannot reliably aggregate — detection of a wrong reference collapses to 9% and clean false alarms rise to 88%. Detection of a buggy evaluator, found by reading code rather than recomputing, holds at 80% either way (Figure 1).

Contributions. (1) A judge-free instrument for measuring LLM benchmark auditors: we construct items where the manifest fixes the correct audit (defect category and witness locus), so an audit is scored as right or wrong without any human or LLM judging the explanation. (2) The dissociation that isolates the cause: only the audit axes that require recomputing the answer collapse on count tasks; the evaluator-code axis, which is found by reading code, does not. Because the control does not move, the collapse is recomputation failure, not general difficulty. (3) The mechanism behind it — the auditor re-solves the task and trusts its own answer over the reference — supported by the dissociation, by verbatim model reasoning, and by a direct probe whose citation pattern reveals what the auditor was actually doing: supplying the verified answer removes the clean false positives and shifts the auditor's citations off the source rows, indicating that the source-localized reference detection observed under baseline rides on re-solving (Section 5). (4) Targeted prompts steer the behavior but do not remove the failure mode (Section 6).

2 Setup and Scoring

A benchmark item is a self-contained task: an *instruction* describing what to do, the *source records* the task is over, a *reference answer* the task is supposed to produce, and an *evaluator* (code that decides whether a candidate answer counts as correct). An auditor reads such an item and decides whether it is clean or whether the defect lies in the instruction, the reference, or the evaluator. We know what is wrong with each item by construction: every item belongs to a *scenario family* (one task and its source records, rendered as four matched items — clean, plus one defect each in the instruction, reference, or evaluator), and a per-item *manifest* records the planted defect's category and the *witness locus* — the artifact lines that make the defect observable. Section 3 describes the construction; we first define the categories and the scoring.

Categories. The four mutually exclusive labels are SPECIFICATION (the instruction omits, weakens, or ambiguously states a load-bearing rule), REFERENCE (the reference answer is inconsistent with the task and source), EVALUATOR (the evaluator does not implement the task), and NONE (no defect; the correct behavior is abstention).

A worked example. Consider a count task: ∼800 source records, N unique qualifying `order_ids`, one of which appears twice. The instruction's load-bearing rule — "keep the earliest row per `order_id`" — gives N; ignoring it gives

$N+1$. The four matched items:

	Instr.	Source	Ref.	Eval.
NONE	rule	dup pair	N	checks N
SPECIFICATION	*deleted*	dup pair	N	checks N
REFERENCE	rule	dup pair	$N{+}1$	checks N
EVALUATOR	rule	dup pair	N	*accepts* $N{+}1$

The mutated cell (italic) is not always the witness: SPECIFICATION's mutation is a *deletion* (no line to cite) and REFERENCE's is just `"answer": `$N{+}1$ (proves nothing alone), so for both the witness is the duplicate pair in the source. EVALUATOR is the asymmetry: its mutated evaluator lines *are* the witness, provable by reading code rather than recomputing N.

Re-solving vs. recomputing. We use *re-solving* for the auditor's general strategy of deriving its own answer from the source rather than checking the reference. On count tasks this entails *recomputing* an aggregate; on other task types it is whatever scan, rank, or projection the answer requires.

Two-axis, judge-free scoring. For non-clean items we score each parse-valid output on two mechanical axes. The *category* axis asks whether the selected category matches the manifest. The *localization* axis asks whether the cited evidence reaches the manifest-recorded *witness locus* — the construction-known artifact lines that make the injected defect observable, defined per category below — not merely whether the citation exists, since here every cited line trivially does ($\sim$100%). Crossing the two axes gives four cells:

TT: correct category and localized — the only success;

TF: correct category, not localized;

FT: wrong category, localized;

FF: wrong category, not localized.

We report **TT** as the result throughout and never report the category axis alone: naming the right category without reaching the witness is not a successful audit. For clean items there is no witness locus, so the output is scored as correct abstention (NONE) or false positive. No step uses a human or LLM judge.

Witness locus, by category. Generalizing the worked example: for SPECIFICATION and REFERENCE the witness is always the source rows that distinguish the correct reading from the planted decoy (the deleted rule has no line to cite; the mutated reference line `[R001]` is ever-present and excluded as boilerplate). For EVALUATOR the witness is the mutated evaluator code lines. Localization therefore asks whether the cited evidence reaches the construction-known witness for that category, not whether it touches the mutated artifact (Appendix A).

Definition 2.1 (Correct audit)**.** For a parse-valid output on a non-clean item, *correct* (TT) means the model selects the manifest category and cites the manifest witness locus. For a clean item, correct means selecting NONE.

3 Construction-Known Items

Our corpus has 200 scenario families, each rendered as four matched items: clean, and one defect each in the instruction (SPECIFICATION), the reference (REFERENCE), and the evaluator (EVALUATOR). An item shows four parts with stable line IDs — the instruction, a source block of 700–877 JSON records, the reference answer, and the evaluator code. Within a family the source block is byte-identical across variants (hash-checked); only the injected defect differs: a load-bearing rule deleted from the instruction, the reference set to a wrong value, or the evaluator mutated to accept the decoy and reject the correct answer. The manifest records each defect's category and witness locus (per category as defined in Section 2; loci listed in Appendix A), and the scorer compares the model's selected category and cited lines against it. Families span three computation mechanisms (dedup, source-precedence, join-key) and six task types (count, single-id, small-list, field-update, url-path, action-target), so results are not tied to one task shape (Appendix A).

The manifest is ground truth. The scorer is deterministic, so what must be validated is the construction. A mechanical gate executes every evaluator and asserts, for all 800 items, that the reference equals an independently-derived answer, the clean variant retains the full rule, the evaluator accepts the correct answer and rejects the decoy, and no construction metadata leaks — all pass. A manual audit of the 20 instruction templates found their rules well-defined, and two annotators inspect 40 stratified items for a present-and-unique defect with the recorded witness locus, confirming all $40/40$ items (Appendix C).

Auditors. We evaluate three production models — Sonnet-4.5, Gemini-2.5-Pro, GPT-4o — under one fixed diagnostic prompt at temperature 0 (Appendix B). Main results are on parse-valid rows (95% overall; Gemini's 87 truncations at the 32k output cap are instrument failures), with an intention-to-treat version in Appendix E.

4 Results

We score 2,400 audits (200 families × four conditions × three models) on parse-valid rows (intention-to-treat in Appendix E).

Table 1 gives the overall rates: correct-audit (TT) is 13% for specification, 57% for reference, and 68% for evaluator defects, and on clean items auditors abstain only 47% of the time. These rates alone could be read as "the auditor is unevenly accurate." The next result shows the structure behind them.

Reference-answer checking is bounded by recomputation. Because each family holds the source records fixed across its four matched items, the EVALUATOR condition is a within-task control: same task, only the artifact slot varies. Partition tasks by whether verifying the answer requires an aggregate the model must compute. *Count* tasks ask for a tally over hundreds of qualifying records; *selection* tasks ask for the top item or top-five, which does not require recomputing an aggregate.

Table 2 is the central result. On count tasks the auditor misses a wrong reference 91% of the time and flags a defect on a clean item 88% of the time, yet its detection of buggy evaluator code — the within-task control, since catching it requires reading code rather than recomputing the answer — is if anything *higher* than on selection tasks. The pattern holds in the same direction for every model (reference TT on count: Sonnet 0%, Gemini 27%, GPT-4o 3%; clean FP on count: 88%/79%/97%; evaluator TT on count: 92%/98%/50%; Section 5.2). The dissociation is the argument: if counting were merely harder, every defect type would be harder to catch. Instead, only the defect types whose verification requires recomputing the answer — a wrong reference, and the abstention decision on a clean item — collapse; the defect type found by reading code does not. The auditor's verdict on the reference is therefore gated by its own ability to compute the answer, not by task difficulty in general.

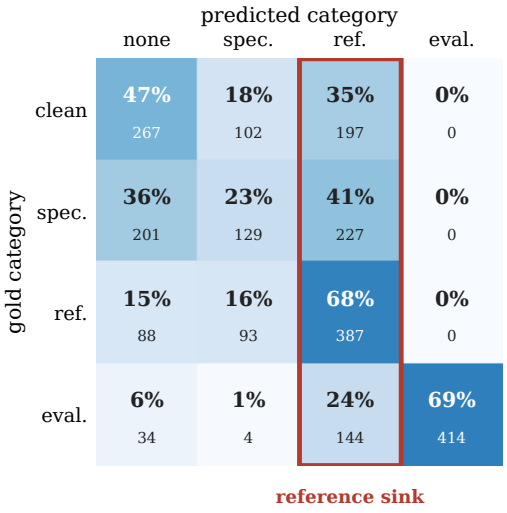

Figure 2: Category confusion (gold × predicted, row-normalized over parse-valid rows). Wrong-category calls concentrate in the REFERENCE column — the reference sink — on clean, specification, and evaluator items alike, while SPECIFICATION↔EVALUATOR confusion is near zero.

The errors route onto the reference. Wrong-category calls concentrate on the reference (Figure 2). Of the 299 clean false positives, 197 (66%) choose REFERENCE versus 102 for SPECIFICATION; committed misdiagnoses of specification and evaluator defects likewise land on REFERENCE, and SPECIFICATION↔EVALUATOR confusion is near zero (per-cell breakdown in Section 5.1). A resolver that disagrees with the reference blames the reference; the direction follows.

Intention-to-treat robustness. The headline does not depend on excluding invalid or truncated rows. Counting all invalid or truncated outputs as failures gives 12.0% TT for SPECIFICATION, 54.0% for REFERENCE, 67.8% for EVALUATOR, and 44.5% clean abstention; the count dissociation is unchanged. The full table is in Appendix E.

4.1 The dissociation is broad, not one task or template

The count collapse is the extreme of a graded pattern: how well the auditor catches a wrong reference tracks how recomputable the answer is (Figure 3). On every task type except count, reference (REFERENCE) and evaluator (EVALUATOR) detection track each other; only on count does reference detection fall to 9% while evaluator detection holds at 80%. The full per-task-type table — with

Condition	n	Category	Localized	TT	TF	FT	FF / FP
SPECIFICATION	557	23% (129)	68% (380)	13% (72)	10% (57)	55% (308)	22% (120)
REFERENCE	568	68% (387)	74% (423)	57% (324)	11% (63)	17% (99)	14% (82)
EVALUATOR	596	69% (414)	77% (457)	68% (407)	1% (7)	8% (50)	22% (132)
NONE	566	abstain: 47% (267)			–		FP: 53% (299)

Table 1: Two-axis decomposition on parse-valid rows. Category asks whether the model selected the manifest category; Localized asks whether its citation reached the manifest witness locus (per category as defined in Section 2). TT (category $\wedge$ localized) is the success metric throughout; we never report Category alone as a result. For clean controls the quantity is abstention vs. false positive. REFERENCE = reference-answer defect; EVALUATOR = evaluator-code defect.

Task group	REFERENCE TT	EVALUATOR TT	Clean FP
count (aggregate)	**9%** (10/108)	80% (96/120)	**88%** (96/109)
selection (rest)	68% (314/460)	65% (311/476)	44% (203/457)

Table 2: The dissociation. On count tasks, detection of a wrong reference (REFERENCE) collapses and false positives on clean items spike, while detection of a buggy evaluator (EVALUATOR), the within-task control, is unaffected — it is found by reading code, not by recomputing the answer.

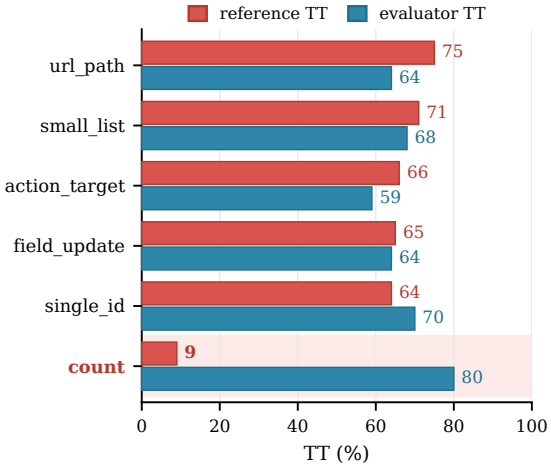

Figure 3: Detection (TT) of a wrong reference (REFERENCE) versus a buggy evaluator (EVALUATOR) by task type. The two track each other except on *count*, where reference detection falls off a cliff (9%) while evaluator detection stays high (80%).

specification TT and clean false positives — is in Appendix G.

Two patterns are visible. First, reference detection and clean abstention fall together as the task's answer gets harder to compute, reaching 9% and 12% on count, while evaluator detection stays 59–80% everywhere — the reference-answer axis is gated by computation, the evaluator axis is not. Second, and distinct from the first, specification TT is low across *all* task types (0–31%), including selection tasks where the answer is easy to compute. This is a re-solving signature that is *not* recomputation-gated: a re-solver resolves a missing instruction rule by its own reading rather than flagging the absence, regardless of task shape. We therefore treat specification blindness as a second, supporting signature of re-solving, separate from the count dissociation — which rests on the reference-vs-evaluator contrast alone.

Across mechanisms. The specification failure is broad across the three construction mechanisms: DEDUP 14% TT, JOIN_KEY 11%, SOURCE_PRECEDENCE 15%; reference and evaluator are much higher for all three (full table in Appendix H). The breadth makes the result hard to dismiss as one idiosyncratic mechanism.

Across templates. The per-template breakdown shows the same: several templates have near-zero specification TT with high evaluator TT — e.g. APPROVAL_EVENTS_USERS (0% vs. 87%), CHECKOUT_EVENTS_CSV (0% vs. 70%), INCIDENT_ACK_EVENTS (0% vs. 97%). The full 20-template table is in Appendix I.

The dissociation survives clustered resampling. We quantify the dissociation with a family-clustered bootstrap (200 families, B=10,000, parse-valid rows), resampling whole families so the four matched variants stay together. The evaluator-minus-reference TT gap is $+70.7$ pp on count (95% CI $[+61.8, +79.4]$) but only -2.9 pp on selection (95% CI $[-7.5, +1.7]$, straddling zero): the gap appears only where the answer must be recomputed. The difference of the two — the dissociation itself — is $+73.7$ pp (95% CI $[+63.7, +83.5]$), so the collapse is specific to count and not a general "evaluator is easier than reference" effect.

5 Mechanism: the auditor re-solves and trusts its own answer over the reference

The dissociation and the directional errors point to one behavior: the auditor re-solves the task from the records and treats any mismatch with the reference as a reference defect. This predicts every result in Section 4: the count collapse (recomputation is least reliable on counts), the clean false positives (an imperfect recomputation disagrees with the correct reference on a clean item), the reference sink (every mismatch is blamed on the reference), and the specification blindness (a re-solver fills a missing rule rather than noticing it is gone).

The models narrate the mechanism. On clean count items — where the reference is, by construction, the correct derived value (Section 3) — Gemini opens with *"I will audit the artifact by re-deriving the correct answer from the source evidence according to the task instructions"* or *"I will manually re-calculate the expected answer,"* produces a tally that is off by one, and concludes the reference is wrong:

> *"The reference answer is 340, but a manual count of qualifying unique order_id values ... yields 341."*
>
> *"The reference answer is 341, but a manual count ... yields 342."*

Both items are clean: 340 and 341 are the construction-derived correct counts, and the model's $+1$ tally is the error that triggers the false "wrong-reference" verdict. The recompute step and the off-by-one that drives the false positive are directly observable. The same stance appears when the defect is real — on a reference-defect count item Gemini writes *"I will audit ... by re-deriving the correct answer ... and comparing it to the reference answer"* and reports a count (338) that is itself wrong: the right verdict for the wrong number.

The same recompute-and-disagree pattern appears across all three models. On the same clean count families Sonnet writes *"let me count records that meet ALL criteria"* and concludes the reference must include records that "should be excluded based on the strict inequality"; GPT-4o writes *"the reference answer of 342 does not match the count of unique* `order_id` *values that meet the specified criteria."* In each case the item is clean and the auditor's recomputation is what disagrees with the correct reference (further excerpts in Section M).

A direct test: remove the re-solving. Removing the need to re-solve should change the behavior predictably. We hand GPT-4o (the purest re-solver) the construction-correct answer on the 32-family subset and tell it not to recompute (the *answer-supplied probe* Probe A′). Clean false positives vanish ($83\%/55\% \rightarrow 0\%$, count/selection) and the auditor stops citing source rows and instead cites the reference line. Because supplying the answer changes the audit task, we do not interpret TT under this condition as an evaluation metric; the citation shift is the mechanism evidence (full discussion Section L), and it indicates that the source-row citations under baseline were a byproduct of re-solving.

Re-solving governs the reference axis, which drives the dissociation. Evaluator failures are a different mode: the model never cites the buggy evaluator lines (the FF cell; Section 5.1) — it fails to inspect the evaluator rather than mis-solving.

5.1 Failure composition

The failure cells in Table 1 are consistent with re-solving, and their internal composition sharpens the picture. We break each failure cell down by what the model did inside it.

Specification fails with the rows in view. Of the 557 parse-valid specification audits, the majority land in FT (308, 55%): the model cites contested source rows but names the wrong category. Where do these go? 156 (51%) call REFERENCE and 152 (49%) abstain; almost none call SPECIFICATION. The FF cell (120, 22%) splits the same way (71 reference, 49 none). In the small TF cell (57, 10%), the model names SPECIFICATION but cites only the ever-present instruction/reference boilerplate (38 of 57) rather than the contested rows.

Caveat. FT for specification shows the contested rows were *in view*, not that the model recognized the ambiguity: those rows are what it cites while re-deriving the answer. We therefore read specification FT as "engaged the relevant evidence and routed the mismatch to the reference," consistent with Section 5 — not as "saw the missing rule and mislabeled it."

Evaluator fails by not inspecting the evaluator code. Evaluator failures are dominated by FF (132, 22%): the model never cites the buggy evaluator lines. Its committed calls in FT/FF go to REFERENCE (82%/78%). This is a localization failure, not a mis-solve — the distinct failure mode noted in Section 5.

Reference is mostly correct. Reference audits are mostly TT (57%); the residual FT (17%) and FF (14%) commit to SPECIFICATION or abstain. The full directional breakdown for all three conditions is in Appendix F.

5.2 The mechanism holds across models

The three models differ only in degree (per-model rates in Table 12); the count over-diagnosis holds for all three (Section 4). GPT-4o is the purest resolver: across parse-valid specification rows it predicts REFERENCE 103 times and NONE 83 times and SPECIFICATION never, while localizing the contested rows 65% of the time — it computes an answer, disagrees with the reference, and blames it or abstains. Renaming the category tokens leaves its specification rate at floor (0.0% → 1.1%), so the failure is behavioral, not a label artifact (Appendix D).

6 Prompt Probes of the Mechanism

The mechanism makes predictions beyond the observational data: a targeted prompt aimed at one putative reason should move the behavior in a specific, predictable way. We test two such prompts on a stratified subset (32 families, 12 count; baseline already collected), each run on the model where the targeted reason is live. Neither is offered as a deployable fix; each is a probe of a reason. Full numbers and methodological notes in Section L.

A: no-mismatch-blame, on GPT-4o. A tells the auditor that disagreeing with the reference on a self-computed answer is not, by itself, a reference defect. It cuts clean false positives (55 → 20% selection, 83 → 75% count) and evaluator→reference misroutes (18 → 5), but true reference detection also falls (60 → 35% selection): a steerable behavior that trades false reference calls for true ones, not a fix.

B: surface-the-rule, on Sonnet. B tells the auditor to check first whether the instruction is missing a load-bearing rule. It roughly doubles how often the model names SPECIFICATION — but on specification (12 → 23) and clean items (12 → 21) alike. Correct specification audits barely move (TT 20 → 25% selection); the extra calls land as TF (named without localization), and clean false positives jump (30 → 65%). Told to look for a missing rule, the model does not differentiate incomplete from complete instructions; it calls SPECIFICATION more often on both. (GPT-4o cannot test B: it never selects SPECIFICATION.)

The mechanism is testable, but neither prompt removes the failure mode.

7 Related Work and Discussion

7.1 Related work

LLM-as-judge, verifiers, and self-verification. LLM-as-judge and reward-model evaluations (Zheng et al., 2023; Gu et al., 2024; Kim et al., 2024), their known biases and calibration gaps (Wang et al., 2024; Liu et al., 2024; Zheng et al., 2024), and self-confidence work (Kadavath et al., 2022) all ask a model to check an answer-bearing artifact rather than produce the answer. The failure we isolate — the verifier re-solves and adjudicates by its own answer — is the central risk for these uses. Benchmark auditing is the setting where we can pin ground truth and score without a judge.

Citation support and attribution. Attribution work asks whether an answer's citations exist, are relevant, and support the claim (Rashkin et al., 2023; Bohnet et al., 2022; Liu et al., 2023), with methods for inline attribution (Asai et al., 2024) and atomic fact-checking (Min et al., 2023). In our closed-context setting existence is trivially satisfied; we score whether the citation reaches the construction-known defect, and find even localized citations accompany wrong diagnoses.

Faithfulness of explanations. An audit is a diagnostic explanation, so our result connects to work on whether model explanations reflect the reasoning that produced them (Wiegreffe et al., 2021; Atanasova et al., 2023; Lyu et al., 2023; Turpin et al., 2023; Lanham et al., 2023): a cited, plausible rationale can accompany a conclusion the evidence does not license.

LLM auditors and benchmark defects. Recent work uses LLMs to audit benchmark items (Tu et al., 2026; Truong et al., 2025), motivated by two threads of documented maintenance failures. The first is task-content defects: underspecified instructions and wrong references in SWE-bench (Jimenez et al., 2024; OpenAI Preparedness Team, 2024; OpenAI, 2026), MMLU (Gema et al., 2025; Zhou et al., 2023), and WebArena (Zhou et al., 2024). The second is train–test contamination (Jain et al., 2025; Deng et al., 2024; Golchin and Surdeanu, 2024). A parallel line strengthens evaluators directly and finds that weak tests inflate

success (Yu et al., 2025; Wang et al., 2026; Yu et al., 2026) — real-world analogs of our reference and evaluator defects. We complement both lines with a construction-known design so the auditor is scored without an LLM judge, following calls for construct-valid capability measurement (Jacobs and Wallach, 2021; Raji et al., 2021; Blodgett et al., 2021) and the behavioral-testing tradition of targeted controlled probes (Ribeiro et al., 2020; Naik et al., 2018; Shankar et al., 2024).

Selective prediction and abstention. The clean false positives connect to abstention work: models fail to withhold when evidence is insufficient (Xiong et al., 2024; Cole et al., 2023; Yin et al., 2023; Varshney et al., 2022; Kamath et al., 2020). Here the over-commitment is downstream of resolving: an imperfect recomputation disagrees with a correct reference and is reported as a defect.

7.2 Discussion

The result identifies a verifier failure mode that may extend beyond benchmark auditing: a model asked to check an answer-bearing artifact can rederive the answer and adjudicate by its own, so its check of the reference answer is bounded by its own task ability. We demonstrate this in benchmark auditing; transfer to other verifier settings (LLM-as-judge, self-verification) is empirically open. This has a direct consequence for deploying LLMs to clean benchmarks: they are least trustworthy exactly where the answer is hardest to compute, and they will emit false "broken reference" reports in proportion to that difficulty — on clean count items, 88%. Because defects are rare in a curated benchmark, this false-positive rate dominates precision: for illustration, at 5% defect prevalence only about 1 flag in 13 is a true defect on a typical task, and 1 in 22 on a count task, so most flags an operator triages are false alarms (Section J). The evaluator axis behaves differently because catching a buggy evaluator requires reading code, not recomputing the answer; this is why a single accuracy number is misleading and why the count dissociation, not the average, is the informative measurement.

8 Conclusion

With a construction-known, judge-free instrument we measured what LLM benchmark auditors actually do. For reference-answer auditing, the models often behave less like independent inspectors and more like re-solvers: they derive their own answer and judge the reference against it. Their checking of the reference answer is therefore bounded by their own task ability: on count tasks, where they cannot reliably tally the records, detection of a wrong reference collapses to 9% (from 68%) and false positives on clean items rise to 88% (from 44%), while detection of buggy evaluator code — found by reading code, not recomputing — is unaffected at 80%. The dissociation, robust across three models and confirmed by the models' own rederivation reasoning, isolates re-computation, not difficulty, as the cause. A direct probe supports the mechanism: supplying the verified answer removes the clean false positives and shifts the auditor's citations off the source rows, so the source-localized reference detection observed under baseline rode on re-solving. We expect this risk in any setting where an LLM verifies an answer-bearing artifact: a check inherits the model's competence on the underlying task, and is least reliable where that task is hardest to compute. The benchmark-audit setting we measure is one such case; whether the same dissociation appears in LLM-as-judge or self-verification is an open empirical question.

Limitations

Our corpus is controlled and template-based — a strength for isolating the mechanism, but the exact rates need not transfer to naturally occurring audits, which future work should test on human-written corpora. We use one fixed prompt at temperature 0 and one run per (sample, model); an abstention-calibrated prompt could lower the clean false positives. The localization axis is manifest-based: a citation can reach the injected region without its quote alone proving the model's explanation, which richer semantic-support annotation would sharpen. The mechanism rests on the dissociation, the verbatim reasoning, and the probe; evaluator failures indicate at least one further mode (not inspecting the evaluator), and the evidence is from one instrument, so transfer to other verifier settings (LLM-as-judge, self-verification) is open. The prompt probes are small-scale: A, Probe A′, and B are each one model on the 32-family subset, and all three are mechanism probes, not deployable remedies.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection.

In *The Twelfth International Conference on Learning Representations (ICLR)*.

Pepa Atanasova, Oana-Maria Camburu, Christina Lioma, Thomas Lukasiewicz, Jakob Grue Simonsen, and Isabelle Augenstein. 2023. Faithfulness tests for natural language explanations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9–14, 2023*, pages 283–294. Association for Computational Linguistics.

Su Lin Blodgett, Gilsinia Lopez, Alexandra Olteanu, Robert Sim, and Hanna Wallach. 2021. Stereotyping Norwegian salmon: An inventory of pitfalls in fairness benchmark datasets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Bernd Bohnet, Vinh Q. Tran, Pat Verga, Roee Aharoni, Daniel Andor, Livio Baldini Soares, Massimiliano Ciaramita, Jacob Eisenstein, Kuzman Ganchev, Jonathan Herzig, Kai Hui, Tom Kwiatkowski, Ji Ma, Jianmo Ni, Lierni Sestorain Saralegui, Tal Schuster, William W. Cohen, Michael Collins, Dipanjan Das, and 3 others. 2022. Attributed question answering: Evaluation and modeling for attributed large language models. *arXiv preprint arXiv:2212.08037*.

Jeremy R. Cole, Michael J. Q. Zhang, Daniel Gillick, Julian Eisenschlos, Bhuwan Dhingra, and Jacob Eisenstein. 2023. Selectively answering ambiguous questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 530–543. Association for Computational Linguistics.

Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. 2024. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 8706–8719. Association for Computational Linguistics.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. 2025. Are we done with mmlu? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 5069–5096. Association for Computational Linguistics.

Shahriar Golchin and Mihai Surdeanu. 2024. Time travel in llms: Tracing data contamination in large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. 2024. A survey on llm-as-a-judge. *CoRR*, abs/2411.15594.

Abigail Z. Jacobs and Hanna Wallach. 2021. Measurement and fairness. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, and 17 others. 2022. Language models (mostly) know what they know. *CoRR*, abs/2207.05221.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5–10, 2020*, pages 5684–5696. Association for Computational Linguistics.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 4334–4353. Association for Computational Linguistics.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamile Lukosiute, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, and 11 others. 2023. Measuring

faithfulness in chain-of-thought reasoning. *CoRR*, abs/2307.13702.

Nelson F. Liu, Tianyi Zhang, and Percy Liang. 2023. Evaluating verifiability in generative search engines. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pages 7001–7025. Association for Computational Linguistics.

Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024. Calibrating llm-based evaluator. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 2638–2656. ELRA and ICCL.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP 2023 -Volume 1: Long Papers, Nusa Dua, Bali, November 1 - 4, 2023*, pages 305–329. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Aakanksha Naik, Abhilasha Ravichander, Norman M. Sadeh, Carolyn P. Rosé, and Graham Neubig. 2018. Stress test evaluation for natural language inference. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 2340–2353. Association for Computational Linguistics.

OpenAI. 2026. Why we no longer evaluate SWE-bench Verified. OpenAI Blog.

OpenAI Preparedness Team. 2024. Introducing SWE-bench Verified. OpenAI Blog.

Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. 2021. AI and the everything in the whole wide world benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.

Hannah Rashkin, Vitaly Nikolaev, Matthew Lamm, Lora Aroyo, Michael Collins, Dipanjan Das, Slav Petrov, Gaurav Singh Tomar, Iulia Turc, and David Reitter. 2023. Measuring attribution in natural language generation models. *Comput. Linguistics*, 49(4):777–840.

Marco Túlio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with checklist. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4902–4912. Association for Computational Linguistics.

Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, J. D. Zamfirescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G. Parameswaran, and Eugene Wu. 2024. SPADE: Synthesizing data quality assertions for large language model pipelines. *Proceedings of the VLDB Endowment*, 17(12):4173–4186.

Sang Truong, Yuheng Tu, Michael Hardy, Anka Reuel-Lamparth, Zeyu Tang, Jirayu Burapacheep, Jonathan Perera, Chibuike Uwakwe, Benjamin Domingue, Nick Haber, and Sanmi Koyejo. 2025. Fantastic bugs and where to find them in AI benchmarks. In *Advances in Neural Information Processing Systems 38: Datasets and Benchmarks Track (NeurIPS)*.

Xinming Tu, Tianze Wang, Yingzhou Lu, Kexin Huang, Yuanhao Qu, and Sara Mostafavi. 2026. BenchGuard: Who guards the benchmarks? Automated auditing of LLM agent benchmarks. *arXiv preprint*, arXiv:2604.24955.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Neeraj Varshney, Swaroop Mishra, and Chitta Baral. 2022. Investigating selective prediction approaches across several tasks in IID, OOD, and adversarial settings. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22–27, 2022*, pages 1995–2002. Association for Computational Linguistics.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. 2024. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9440–9450. Association for Computational Linguistics.

You Wang, Michael Pradel, and Zhongxin Liu. 2026. Are "solved issues" in SWE-bench really solved correctly? an empirical study. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering (ICSE)*.

Sarah Wiegreffe, Ana Marasović, and Noah A. Smith. 2021. Measuring association between labels and

free-text rationales. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Online and Punta Cana, Dominican Republic, November 7–11, 2021*, pages 10266–10284. Association for Computational Linguistics.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2024. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. 2023. Do large language models know what they don't know? In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Findings of ACL, pages 8653–8665. Association for Computational Linguistics.

Boxi Yu, Yang Cao, Yuzhong Zhang, and 1 others. 2026. SWE-ABS: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark. ArXiv:2603.00520.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. 2025. UTBoost: Rigorous evaluation of coding agents on SWE-Bench. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2024. Judging llm-as-a-judge with MT-bench and chatbot arena (placeholder). Placeholder citation – author should select the intended "evaluation of LLM judges" paper before submission. [VERIFY-AUTHORS] Cite key chosen by the paper's authors but exact paper not specified; pick the intended reference and replace this entry.

Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. 2023. Don't make your LLM an evaluation benchmark cheater. *CoRR*, abs/2311.01964.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Construction Details

The corpus is synthesized by a single deterministic generator. There are 20 templates $\times$ 10 seeds $= 200$ scenario families; the per-family seed is a fixed function of the template index and seed index, so the corpus regenerates byte-for-byte. For each family the generator first builds one *source case* — the source records plus the family's correct and decoy answers — and then renders that one source case into the four variants (NONE, SPECIFICATION, REFERENCE, EVALUATOR). Because the four variants share the same source case, the SOURCE block is identical across them by construction (and verified by hash in `validate.py`); only the injected defect differs.

Mechanisms and the load-bearing rule. Every family is built around one computation mechanism whose rule the auditor must apply to get the right answer, and a planted decoy that yields a specific wrong answer if the rule is ignored.

- **DEDUP.** Qualifying rows can repeat an entity id; the rule is to keep the earliest `row_id` as the canonical row per id. The decoy is a second, higher-priority row for one already-present id (the witness pair). Applying the rule deduplicates it away; ignoring the rule counts/selects it as a separate unit.
- **SOURCE_PRECEDENCE.** Each entity may appear in a `primary` and a `fallback` source; the rule is to ignore `fallback` rows for any entity that has a `primary` row in the window. The decoy is a conflict entity whose `primary` row has a non-qualifying status and whose `fallback` row qualifies: under the rule the entity is excluded, ignoring precedence includes it.
- **JOIN_KEY.** Events name an actor by both id and display name; the rule is to join on the actor id and never treat the display name as an identifier. The decoy is two users who share a display name — one with the qualifying role, one without — and an event by the non-qualifying user: joining by id excludes the event, matching on the shared name includes it.

In all three the *correct* answer applies the rule and the *decoy* answer is the plausible rule-violating reading; the generator asserts the two differ for every family. A row qualifies only if it matches the family's scope/status/source predicates and falls in the April window; several hundred non-qualifying noise rows (wrong scope, wrong status, wrong source, out-of-window times) are seeded so the task requires discrimination, not just a scan.

Task types and rendering. The selected units are rendered into one of six output shapes: `count` (number of qualifying units), `single_id` (top-ranked unit), `small_list` (top five), `field_update` (an update object for the top unit), `url_path` (a URL built from the top unit), and `action_target` (the top unit's target id). Ranking is by descending `priority`, ties broken by `row_id`. The 20 templates tile the mechanisms and task types:

- **DEDUP** (6): `checkout_events_csv` (count), `support_ticket_log` (single_id), `invoice_line_items` (small_list), `deployment_events` (field_update), `search_result_cards` (url_path), `catalog_cards_website` (action_target).
- **SOURCE_PRECEDENCE** (6): `order_status_snapshots` (count), `policy_control_plane_trace` (single_id), `entitlement_state` (small_list), `moderation_state` (field_update), `feature_flag_state` (url_path), `inventory_live_cache_website` (action_target).
- **JOIN_KEY** (8): `approval_events_users` and `incident_ack_events` (count), `lms_submission_events` (single_id), `repo_review_events` and `calendar_approval_events` (small_list), `crm_owner_activity` (field_update), `warehouse_pick_events` (url_path), `admin_action_website` (action_target).

Defect injection. Each variant is the shared source case rendered with exactly one change. NONE: the instruction states the full rule, the reference equals the correct answer, and the evaluator checks the prediction against the correct answer. SPECIFICATION: the single load-bearing rule sentence is deleted from the instruction; the reference and evaluator are unchanged (still correct), so the only defect is the missing rule. REFERENCE: the reference value is replaced by the decoy answer; the instruction and evaluator are unchanged. EVALUATOR: the clean equality evaluator is replaced

by one that recomputes the answer with the rule-violating selection step (so it accepts the decoy and rejects the correct answer); the instruction and reference are unchanged. The mutated evaluator lines are recorded as the witness locus.

Item layout and line IDs. A rendered item is a newline-delimited block in which every line carries a stable bracketed ID: `[I001]` the instruction; `[S001]` a source header followed by `[C00001]`... records (700–877 JSON rows); `[R001]` the reference value and `[R002]` a fixed derivation note; and `[K001]` a evaluator header followed by `[G001]`... code lines. This is the artifact pasted verbatim into the prompt (Section B).

Witness loci. The manifest records, per item, the artifact lines a correct citation must reach — the *witness locus*. For SPECIFICATION the load-bearing rule is deleted, not mutated, so there is no instruction line to cite as the defect; the witness is the source rows that make the decoy wrong only under the missing rule (e.g. the DEDUP duplicate pair) plus, for non-count tasks, the *target* rows that the correct and decoy readings select. For REFERENCE the mutated reference line ([R001]) is ever-present across all four variants and so excluded from the localization axis as boilerplate; the witness is again the source rows that distinguish the correct answer from the injected decoy. For EVALUATOR the witness is the mutated evaluator code group. Because the ever-present `[I001]` and `[R001]` lines are excluded as boilerplate, citing them does not count as reaching the witness; this matters specifically for SPECIFICATION and REFERENCE (whose mutated artifact lines fall in this excluded set) and is the reason the witness for those categories is defined on the source rows rather than the mutated artifact.

No construction leakage. Three measures keep the construction out of the visible artifact, all checked by `validate.py`. (i) The internal bookkeeping keys (the witness tag and the line-ID field) are stripped from each record before it is serialized, so the visible JSON carries only task fields. (ii) Randomly generated ids are rejected if they contain any helper token (e.g. `WITNESS`, `DUP`, `SHARED`), so no id hints at its role. (iii) The witness rows are scattered to deterministic but non-adjacent positions (near the quarter, seven-eighths, midpoint, third, and end of the block) rather than grouped, so they are not visually flagged.

Worked example (DEDUP count). In a `checkout_events_csv` family the source contains N unique qualifying `order_ids`, one of which appears twice: a second, higher-priority row for the same order (the witness pair), buried among the records. The rule — keep the earliest `row_id` as canonical per `order_id` — gives the correct `completed_order_count` of N; ignoring it counts the duplicate and gives $N + 1$. The four variants of this family are then: NONE reference $= N$ with a evaluator that checks for N; SPECIFICATION identical but with the dedup sentence removed from the instruction; REFERENCE reference set to $N + 1$; EVALUATOR a evaluator whose selection step omits the dedup grouping, so it computes $N + 1$ and rejects the correct N. This is the family behind the verbatim off-by-one in Section 5, where the auditor recounts, fails to deduplicate, and reports $N + 1$ against a correct reference of N.

B Prompts and Output Schema

Every model sees the same fixed diagnostic prompt, reproduced verbatim in Figure 4. The only edits are hard line wraps to fit the page (marked by indentation); the token names, category descriptions, and instructions are exact. The `Categories` block shown is the baseline variant; the label-robustness check (Section D) substitutes the four token names listed in Table 3 and leaves everything else — descriptions, ordering, and the rest of the prompt — unchanged.

Output schema. The audit is read from the final `FINAL_JSON` line, with four fields: `primary_category` (one of the four category tokens) drives the category axis; `citations` (a list of at most 8 line-ID strings) drives the localization axis; `finding` is the free-text rationale; and `confidence` is recorded but not used in scoring. A row is parse-valid when this final line is present and yields a category token and a citation list; rows that omit it or truncate before it are invalid (Section 3).

```
You are auditing one benchmark task artifact.

The artifact may be clean or may contain a benchmark defect.

Categories:
- none: no benchmark defect is supported by the artifact
- specification: the task instruction is ambiguous, contradictory,
  or missing a load-bearing rule
- oracle: the reference answer is wrong relative to the task and
  source evidence
- evaluator: the grading code accepts an incorrect prediction or
  rejects a correct prediction

Use only the visible artifact. Do not assume access to any hidden
answer key.

Every artifact line has a stable line ID in square brackets, such as
[I001], [R001], [G001], or [C00001]. If you claim a defect, cite the
smallest set of line IDs needed to prove the finding. Use at most 8
citations.

Return free reasoning if useful, but the final line must be exactly:
FINAL_JSON: {"primary_category": "...", "confidence": 0.0,
"finding": "...", "citations": ["..."]}

VISIBLE BENCHMARK ARTIFACT:
<[I...] instruction; [C...] ~800 source records;
 [R...] reference answer; [G...] grader code>
```

Figure 4: The fixed diagnostic prompt sent to all three auditor models, reproduced verbatim. Hard line wraps (marked by indentation in the `Categories` block) are the only edits.

Gold category	Baseline token	Robustness token
NONE	`none`	`clean`
SPECIFICATION	`specification`	`instruction`
REFERENCE	`oracle`	`reference`
EVALUATOR	`evaluator`	`grader`

Table 3: Category tokens under the two label variants. The label-robustness check (Section D) swaps the baseline tokens for the neutral artifact words; the descriptions are byte-identical across variants, and the manifest maps the robustness tokens back to the gold categories before scoring.

C Item (Construction) Audit

The scorer is deterministic, so the human check targets the construction, not the auditor: it confirms the manifest is the true ground truth. Two annotators inspect a stratified sample of 40 items (10 each for clean, specification, reference, evaluator; spread across the three mechanisms and six task types, with count tasks included). For a clean item the annotator confirms no defect is present (instruction complete, reference correct under the stated rule, evaluator correct); for a defect item, that the injected defect is present at the recorded witness locus and is the *only* defect. This complements the mechanical gate (Section 3), which already executes every evaluator and checks reference/decoy behavior on all 800 items. Both annotators confirmed all $40/40$ items (full agreement; no items flagged for revision).

D GPT-4o Label-Robustness Check

We re-run GPT-4o with the four category tokens renamed to neutral artifact words (`instruction`/`reference`/`grader`/`clean`), with descriptions unchanged, and map them back (`instruction`→SPECIFICATION, etc.) before scoring. Table 4 gives the per-condition before/after. The specification cell — the only place the label-vocabulary objection was load-bearing — stays at floor: category $0.0\% \rightarrow 1.1\%$, TT $0.0\% \rightarrow 0.6\%$, with localization essentially unchanged ($64.5\% \rightarrow 61.3\%$). The renamed `instruction` token is reachable: GPT-4o emits it on 3 parse-valid rows (2 specification, 1 reference), so the model is able to use the label and simply almost never does. The rename is not perfectly neutral on other conditions (reference TT rises, clean false positives rise), but the targeted question is answered: label vocabulary does not explain GPT-4o's specification failure.

Condition	Variant	p-valid	Cat/Abst	Loc	TT
NONE	before	194/200	40.2%	–	–
NONE	after	192/200	27.6%	–	–
SPECIFICATION	before	186/200	0.0%	64.5%	0.0%
SPECIFICATION	after	181/200	1.1%	61.3%	0.6%
REFERENCE	before	195/200	74.4%	64.1%	53.3%
REFERENCE	after	195/200	81.5%	66.2%	61.5%
EVALUATOR	before	199/200	16.6%	33.7%	15.6%
EVALUATOR	after	195/200	12.8%	37.9%	11.8%

Table 4: GPT-4o before/after renaming the category tokens. Cat/Abst is the category rate for non-clean conditions and the abstention rate for NONE. The specification floor survives the rename.

Condition	TT	TF	FT	FF	inv.
SPECIFICATION	12.0%	9.5%	51.3%	20.0%	7.2%
REFERENCE	54.0%	10.5%	16.5%	13.7%	5.3%
EVALUATOR	67.8%	1.2%	8.3%	22.0%	0.7%
NONE	abstain		FP		inv.
	44.5%		49.8%		5.7%

Table 5: Intention-to-treat decomposition. Denominator is all 600 attempted audits per condition; invalid or truncated outputs (column *inv.*) are counted as failures rather than excluded. Underlying counts: SPECIFICATION TT $72/600$, REFERENCE TT $324/600$, EVALUATOR TT $407/600$, NONE abstention $267/600$. The headline TT rates and the count dissociation are unchanged relative to the parse-valid analysis.

E Intention-to-Treat Results

The main-paper tables score parse-valid rows. Here we repeat the per-condition decomposition under an intention-to-treat (ITT) convention: every attempted audit is in the denominator (600 per condition = 200 families $\times$ 3 models), and any invalid or truncated output is counted as a failure rather than dropped. The only non-parse-valid rows that are not ordinary parse failures are Gemini-2.5-Pro's 87 truncations at the 32k output cap (Section 3); under ITT these are charged as failures.

Table 5 gives the result. The headline TT rates — specification 12.0%, reference 54.0%, evaluator 67.8%, clean abstention 44.5% — track the parse-valid rates closely (cf. Table 1), confirming that excluding invalid and truncated rows does not drive the reported pattern.

F Failure Direction Breakdowns

Table 6 expands the three failure cells of Table 1 by what the model did inside each cell, on parse-valid rows. For the off-diagonal cells (FT, FF) the direction is the category the model named ("abstain" = predicted NONE); for TF (right category, not localized) it is what the model cited instead of the witness locus. Boilerplate refers to the ever-present `I001/R001` lines excluded from the localization axis.

The reference sink is also visible model-by-model in the specification misroute (Table 7): committed specification errors go to REFERENCE, never to EVALUATOR, and almost always cite the reference line.

Condition	Cell	n	Within-cell direction
SPECIFICATION	TF	57	boilerplate 38; other non-locus 19
	FT	308	REFERENCE 156 (51%); abstain 152 (49%)
	FF	120	REFERENCE 71 (59%); abstain 49 (41%)
REFERENCE	TF	63	other non-locus 61; boilerplate 2
	FT	99	SPECIFICATION 55 (56%); abstain 44 (44%)
	FF	82	abstain 44 (54%); SPECIFICATION 38 (46%)
EVALUATOR	TF	7	other non-locus 7
	FT	50	REFERENCE 41 (82%); abstain 6 (12%); SPECIFICATION 3 (6%)
	FF	132	REFERENCE 103 (78%); abstain 28 (21%); SPECIFICATION 1 (1%)

Table 6: Within-cell failure directions by condition (parse-valid). When a non-reference defect (SPECIFICATION, EVALUATOR) fails into a committed call it routes to REFERENCE; REFERENCE failures route to SPECIFICATION. SPECIFICATION↔EVALUATOR confusion is near zero.

Model	SPECIFICATION→REFERENCE	SPECIFICATION→EVALUATOR	cites reference
Sonnet-4.5	51	0	100% (51/51)
Gemini-2.5-Pro	73	0	100% (73/73)
GPT-4o	103	0	97% (100/103)

Table 7: Specification misroute by model: committed specification errors land on REFERENCE (never EVALUATOR) and cite the reference line.

G Task-Type Breakdown

Task type	SPECIFICATION TT	REFERENCE TT	EVALUATOR TT	Clean FP
action_target	31% (27/88)	66% (59/89)	59% (53/90)	52% (46/88)
count	**0% (0/100)**	**9% (10/108)**	80% (96/120)	**88% (96/109)**
field_update	13% (11/85)	65% (56/86)	64% (58/90)	45% (39/87)
single_id	17% (15/90)	64% (57/89)	70% (61/87)	47% (42/89)
small_list	8% (9/108)	71% (79/112)	68% (82/120)	35% (38/108)
url_path	12% (10/86)	75% (63/84)	64% (57/89)	45% (38/85)

Table 8: TT by task type and condition, plus clean false-positive rate. Reference detection and clean abstention degrade as the answer becomes harder to recompute, bottoming out on count; evaluator detection (code reading) stays high throughout. Specification TT is low across all task types.

H Mechanism Breakdowns

Table 9 gives TT by condition and clean abstention for each of the three construction mechanisms (parse-valid). Specification TT is low for all three (11–15%) while reference and evaluator are much higher, so the specification failure is not an artifact of one mechanism.

Mechanism	SPECIFICATION TT	REFERENCE TT	EVALUATOR TT	Clean abstain
DEDUP	14% (24/176)	57% (101/176)	64% (116/180)	41% (73/176)
JOIN_KEY	11% (23/210)	55% (117/213)	74% (175/237)	54% (115/213)
SOURCE_PRECEDENCE	15% (25/171)	59% (106/179)	65% (116/179)	45% (79/177)

Table 9: TT by mechanism and condition, plus clean abstention (parse-valid). Specification TT is uniformly low across mechanisms; reference and evaluator are uniformly higher.

I Template Breakdowns

Table 10 gives TT by condition and clean abstention for all 20 templates (parse-valid). Several templates have near-zero specification TT with high evaluator TT (e.g. APPROVAL_EVENTS_USERS 0% vs. 87%, INCIDENT_ACK_EVENTS 0% vs. 97%), and the count-task templates (CHECKOUT_EVENTS_CSV, ORDER_STATUS_SNAPSHOTS) carry the reference collapse.

Template	SPECIFICATION TT	REFERENCE TT	EVALUATOR TT	Clean abstain
ADMIN_ACTION_WEBSITE	34% (10/29)	55% (16/29)	60% (18/30)	34% (10/29)
APPROVAL_EVENTS_USERS	0% (0/27)	4% (1/27)	87% (26/30)	18% (5/28)
CALENDAR_APPROVAL_EVENTS	4% (1/27)	66% (19/29)	77% (23/30)	78% (18/23)
CATALOG_CARDS_WEBSITE	33% (10/30)	83% (25/30)	57% (17/30)	53% (16/30)
CHECKOUT_EVENTS_CSV	0% (0/26)	4% (1/26)	70% (21/30)	4% (1/27)
CRM_OWNER_ACTIVITY	8% (2/25)	88% (23/26)	70% (21/30)	70% (19/27)
DEPLOYMENT_EVENTS	7% (2/30)	53% (16/30)	60% (18/30)	47% (14/30)
ENTITLEMENT_STATE	7% (2/28)	77% (23/30)	67% (20/30)	40% (12/30)
FEATURE_FLAG_STATE	13% (4/30)	80% (24/30)	60% (18/30)	53% (16/30)
INCIDENT_ACK_EVENTS	0% (0/23)	4% (1/26)	97% (29/30)	20% (5/25)
INVENTORY_LIVE_CACHE_WEBSITE	24% (7/29)	60% (18/30)	60% (18/30)	55% (16/29)
INVOICE_LINE_ITEMS	17% (5/30)	60% (18/30)	63% (19/30)	59% (17/29)
LMS_SUBMISSION_EVENTS	20% (6/30)	69% (20/29)	68% (19/28)	57% (17/30)
MODERATION_STATE	23% (7/30)	57% (17/30)	63% (19/30)	50% (15/30)
ORDER_STATUS_SNAPSHOTS	0% (0/24)	24% (7/29)	67% (20/30)	7% (2/29)
POLICY_CONTROL_PLANE_TRACE	17% (5/30)	57% (17/30)	72% (21/29)	62% (18/29)
REPO_REVIEW_EVENTS	4% (1/23)	83% (19/23)	67% (20/30)	88% (23/26)
SEARCH_RESULT_CARDS	10% (3/30)	70% (21/30)	67% (20/30)	43% (13/30)
SUPPORT_TICKET_LOG	13% (4/30)	67% (20/30)	70% (21/30)	40% (12/30)
WAREHOUSE_PICK_EVENTS	12% (3/26)	75% (18/24)	66% (19/29)	72% (18/25)

Table 10: TT by template and condition, plus clean abstention (parse-valid), for all 20 templates.

J Prevalence-Sensitive Precision

An operator who runs the auditor over a benchmark does not see TT or false-positive rates directly; they see flags and must decide which to trust. The relevant quantity is positive predictive value (PPV) — given a raised flag, the probability that a defect is actually present:

$$\mathrm{PPV}(p) = \frac{p \cdot \mathrm{sens}}{p \cdot \mathrm{sens} + (1-p) \cdot \mathrm{FPR}},$$

where p is the prevalence of defective items in the pool, sens is the rate at which the auditor flags a defect when one is present, and FPR is the rate at which it flags a defect on a clean item. We take a "flag" to be any non-NONE category call (the event an operator triages, regardless of whether the category is ultimately correct). From the parse-valid data, $\mathrm{sens} = 1398/1721 = 81\%$ (the pooled defect-call rate on defect items) and $\mathrm{FPR} = 299/566 = 53\%$ overall, rising to 88% on count tasks (Table 2).

Table 11 sweeps prevalence. At the low defect rates a curated benchmark actually carries, PPV is dominated by the $(1-p)\,\mathrm{FPR}$ term and stays low even though sensitivity is high: at 5% prevalence PPV is 7.5% on a typical task and 4.6% on a count task, so the great majority of flags are false alarms. We isolate the count effect by holding sensitivity fixed and moving only the FPR, which is legitimate here because the defect-call sensitivity does not collapse on count — evaluator detection is intact (Table 2); only the clean false-positive rate moves.

Prevalence p	PPV, all tasks (FPR 53%)	PPV, count tasks (FPR 88%)
1%	1.5%	0.9%
2%	3.0%	1.8%
5%	7.5%	4.6%
10%	14.6%	9.3%
20%	27.8%	18.7%

Table 11: Positive predictive value of a raised flag as a function of defect prevalence, holding sensitivity at the pooled 81% and varying only the clean false-positive rate (the term that dissociates between task groups). Because defects are rare in a curated benchmark, most flags are false: at 5% prevalence only $\sim$1 in 13 flags is real on a typical task, and $\sim$1 in 22 on a count task.

Requiring more of a flag lowers PPV further. If a flag counts as a true positive only when it names the correct category, the numerator sensitivity falls to $930/1721 = 54\%$ (a factor 0.66); if it must also be localized (TT), to $803/1721 = 47\%$ (a factor 0.57). At 5% prevalence these give PPV 5.0% and 4.3% respectively. The modeling assumptions are deliberately simple: one defect per defective item, a single prevalence across types, and a swept p because the true defect rate of a target benchmark is unknown but generally small.

K Per-Model Results

Model	SPECIFICATION TT (cat/loc)	REFERENCE TT (cat/loc)	EVALUATOR TT (cat/loc)	Clean abstain
Sonnet-4.5	14% (27/200; 30%/68%)	57% (114/200; 62%/79%)	94% (187/200; 94%/100%)	54% (109/200)
Gemini-2.5-Pro	26% (45/171; 40%/73%)	61% (106/173; 68%/81%)	96% (189/197; 98%/97%)	47% (80/172)
GPT-4o	0% (0/186; 0%/65%)	53% (104/195; 74%/64%)	16% (31/199; 17%/34%)	40% (78/194)

Table 12: Per-model results by condition (parse-valid); parentheses give TT count and category/localization rates. GPT-4o is the purest re-solver — it behaves almost as a reference-checker, never naming SPECIFICATION and catching evaluator defects rarely on selection tasks.

GPT-4o subset (fixed denom.)	baseline	Probe A
clean FP, selection (lower better)	55%	20%
clean FP, count	83%	75%
reference TT, selection (guard)	60%	35%
evaluator TT, selection	0%	20%
evaluator TT, count	42%	67%
eval→reference misroutes (committed)	18	5

Table 13: Probe A (no-mismatch-blame prompt) on the GPT-4o subset. A cuts reference-blaming (clean FP and misroute volume) but lowers true reference detection (the guard), so the prompt nudge suppresses reference calls rather than correcting attribution.

GPT-4o subset (fixed denom.)	baseline	Probe A′
clean FP, count (lower better)	83%	0%
clean FP, selection	55%	0%
names REFERENCE on reference, count	83%	100%
names REFERENCE on reference, selection	65%	100%
reference TT, count	8%	0%
reference TT, selection	60%	35%
eval→reference misroutes (committed)	100%	0%

Table 14: Answer-supplied probe (Probe A′) on the GPT-4o subset. Handing the model the verified answer eliminates false positives and the reference sink and makes category detection perfect, but true detection (TT) does not recover: localization collapses because the model no longer engages the source it would have computed over. The probe confirms the mechanism (Section 5); it is not a deployable fix.

L Probe Details

Table 13 and Table 14 give the full before/after for the two GPT-4o prompt probes on the 32-family subset (fixed denominators): the no-mismatch-blame probe A (Section 6) and the answer-supplied probe Probe A′ (Section 5). Both are mechanism probes, not deployable fixes.

Why Probe A′ TT does not measure detection. Under Probe A′ the audit task itself has changed: once the verified answer is supplied, the reference line plus that answer is in principle sufficient to back a reference-defect call, so the baseline TT criterion (which requires citing the source rows that distinguish correct from decoy) is no longer the right measure of correctness. We therefore read Probe A′ only as evidence that the reference axis under baseline — the clean false positives and the source-row citations — is downstream of resolving, not as an evaluation of answer-assisted auditing.

M Verbatim Reasoning Excerpts

Section 5 quotes Gemini-2.5-Pro on clean CHECKOUT_EVENTS_CSV items where the reference is, by construction, the correct count N and the auditor's recomputation produces $N+1$. The same recompute-and-disagree pattern appears in Sonnet-4.5 and GPT-4o on clean count items; below are representative excerpts (quotes minimally trimmed for length; line citations preserved where present).

Sonnet-4.5, clean CHECKOUT_EVENTS_CSV (count, N=340).

> *"... let me count records that meet ALL criteria including the correct date range... The reference answer appears to incorrectly include records with* `event_time='2026-05-01T00:00:00Z'`*... If these records are included in the count of* 340*, the reference answer is incorrect."*

By construction the clean reference is 340; Sonnet's recomputation differs and the disagreement is reported as a reference defect.

GPT-4o, clean CHECKOUT_EVENTS_CSV (count, N=342).

> *"The reference answer of* 342 *does not match the count of unique* `order_id` *values that meet the specified criteria."*

Again the item is clean: 342 is the correct construction-derived count; the model's recomputation is the error.

GPT-4o, clean APPROVAL_EVENTS_USERS (count, N=184).

> *"The reference answer count of* 184 *does not match the actual count of qualifying events based on the task specification and provided data."*

The phrase *"the actual count"* is the model's own recomputed value, treated as the standard against which the (correct) reference is judged.

Pattern across models. In each excerpt the auditor (i) recomputes the count from the source records, (ii) observes a discrepancy with the reference, and (iii) attributes the discrepancy to the reference. Because the items are clean, the discrepancy is the auditor's own counting error.