WHEN METRICS MISLEAD: AUDITING THE MEASUREMENT LAYER OF LLM AGENT EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are often evaluated by reducing execution traces to labels such as final-answer correctness, tool verification, and fault propagation. These labels are useful, but they are measurements produced by parsers, behavioral detectors, and harness rules. We study what happens when those measurements do not fully capture the trace's behavior. We use GSM8K, a benchmark of grade-school arithmetic word problems, in a tool-use setting where agents can call a calculator whose outputs are occasionally corrupted. This gives us a clear ground truth for the expected tool result and allows us to compare what the calculator returned, what the model later used, and what the automated detectors reported. We audit three trace-level detectors across six production model families and 30 problems per model. We find that final correctness, visible re-verification, and fault propagation can disagree in systematic ways. An important failure mode is the silent tool-output override. In these cases, the calculator first returns a corrupted value, but the model later uses a different, often correct, value without calling the calculator again. The bad value may therefore stop propagating even though a verification detector that looks for explicit re-execution records no visible re-check. Human review of 90 traces shows perfect agreement with the reviewed comparable cases for final correctness (67/67), fault recovery/impact (19/19), and fault relevance (18/18), but only 67/81 agreement for verification. All 14 verification disagreements fall into four recurring patterns: repeated calculator calls from stuck loops, verbal recognition of an incorrect tool result without recomputation, repeated expressions within a single reasoning step, and quiet correction after an incorrect value is observed. We also find that execution limits affect the reported result. In an ablation study across two models, one model ranges from 21/30 to 27/30 correct as step and token budgets change, while another remains between 27/30 and 29/30. These findings motivate measurement transparency: an automated label should be accompanied by trace evidence supporting it, whether the detector's question applied to that trace, whether the run ended early due to an execution limit, and any human corrections to the label. Our trace viewer makes this information inspectable alongside the reported results.

1 INTRODUCTION

Agent evaluation is a measurement process. An agent produces an execution trace–a record of its model and tool interactions–which a parser or detector maps to a label. Labels are then aggregated into numbers such as accuracy, verification rate, or fault-propagation rate. Those numbers are often treated as facts about the agent. In practice, they also depend on the instruments used to measure the agent.

This distinction matters because agents are increasingly evaluated for tool use and multi-step decision-making. In such settings, a small measurement mistake can change the interpretation of a run. A trace may contain a correct answer that the parser does not extract. A detector may ask a question that is not meaningful for a particular event. A step limit may stop an agent before it finishes, but the benchmark may still count the run as simply wrong. In higher-stakes settings, such as clinical, defense, or other safety-critical workflows, this kind of false confidence is especially dangerous: an agent can look most trustworthy exactly when it is quietly overriding its instruments.

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

Knowing that an agent received a score is not enough; we also need to know whether the score reflects a trustworthy measurement.

Recent work has already shown that the reliability of the evaluator cannot be assumed. Agent benchmark audits find disagreement between automated evaluators and human annotators, tool-calling benchmarks contain brittle judging rules, and agent performance can depend on the harness in which the model is run (Bean et al., 2025; Gurram, 2026; Bhat et al., 2026; Yao et al., 2026). We build on that line of work, but focus on a narrower question: when an automated trace detector reports a label, did the trace actually contain the right evidence for that label to be interpreted safely?

We study the question in a deliberately simple domain. Agents solve arithmetic word problems using a calculator tool. The calculator sometimes returns a plausible but wrong value. Because arithmetic has external ground truth, we can check four things separately: what the calculator should have returned, what it actually returned, what value the model later used, and what final answer the model gave. This makes the setting useful for auditing the measurement apparatus itself.

The motivating failure is silent tool-output override. A model receives a corrupted calculator result, but later uses a different value, often the independently correct one, without calling the calculator again and without explicitly stating that the tool was wrong. Prior work on tool-memory conflict already shows that models arbitrate between internal knowledge and tool outputs (Cheng et al., 2025; Varlamov et al., 2026). Our claim is not that this behavior is newly discovered, but that it creates a measurement problem. For example, if the calculator returns 29 for a quantity whose correct value is 30 and the model later continues its reasoning with 30 without another tool call, a propagation detector may correctly record that 29 was not reused while a verification detector still reports no visible re-check. The trace therefore contains evidence of recovery that the verification detector was not designed to represent.

Contributions.

- **C1. Measurement transparency for trace detectors.** We define a practical reporting principle for agent evaluation: a detector label should be accompanied by the evidence that supports it, whether the detector was applicable, whether execution was censored, and any known measurement risks or human corrections.
- **C2. A controlled audit of detector reliability.** Across six model families, we show that final correctness, visible verification, and fault propagation can diverge in systematic ways, including silent override, inconclusive propagation, formatting failures, and harness-dependent censoring. We also compare automated labels with human review.
- **C3. A trace viewer that reveals the measurement behind the score.** We build a reliability-aware visualization that links aggregate results to the trace evidence behind each detector output and preserves automatic labels together with human corrections. This makes it possible to inspect why a measurement should or should not be trusted rather than treating the final score as self-explanatory.

2 Related Work

Evaluator validity. Construct-validity work argues that a benchmark score is meaningful only when the task and metric support the claim being made from them (Bean et al., 2025). Recent audits make this concern concrete for agents. AgentProp-Bench compares automated judging protocols with human labels and studies error propagation in tool-using agents (Gurram, 2026). Bhat et al. (2026) audit several tool-calling benchmark families and report evaluator-human disagreements, brittle deterministic rules, judge variance, and corrected benchmark components. REFLECT similarly meta-evaluates LLM judges using controlled interventions on agent traces and finds substantial failure in fine-grained failure detection (Wang et al., 2026). AgentJudgeBench studies judge reliability specifically for structured tool-calling workflows (Verma et al., 2026). Our work is complementary: rather than evaluating an LLM judge as a whole, we focus on behavioral trace detectors and ask whether a detector had the observable and applicable evidence needed to support its label.

Trace- and process-level evaluation. Recent work increasingly evaluates how an agent reaches an outcome rather than only whether it succeeds. ClawTrack separates task success from process quality and includes result verification as a process dimension (Wu et al., 2026). CTF-ABACUS

reconstructs agent trajectories to distinguish demonstrated exploitation from shortcut or unsupported success in cybersecurity tasks (Milner et al., 2026). These works reinforce the value of trace-level evidence. Our question is one level lower: whether the measurement applied to that trace faithfully represents the behavior it is intended to measure.

Tool-output conflict. Tool-Memory Conflict and related work study cases where model knowledge and tool output disagree (Cheng et al., 2025; Varlamov et al., 2026). We do not rename that phenomenon or claim to discover it. Instead, we ask what happens when such conflict appears inside an evaluation pipeline. A model may recover from a bad tool value in a way that improves the final answer while still failing the detector's narrow definition of verification.

Harness effects. Harness-Bench shows that agent performance should often be reported at the model-harness configuration level rather than as a property of the model alone (Yao et al., 2026). Here, the evaluation harness–the runtime configuration governing execution limits and tool behavior—matters because step and token limits can censor traces, while per-call fault injection gives different fault exposure to agents that make different numbers of tool calls. These are not just engineering details; they change what the reported denominator means.

Trace visualization. Systems such as AgentLens and Graph of Trace make agent behavior easier to inspect (Lu et al., 2024; Gao et al., 2026). Agentic CLEAR performs system-, trace-, and node-level agent evaluation and presents the resulting analysis through an interactive interface (Yehudai et al., 2026). Bhat et al. (2026)'s Harness Lab likewise supports trace-level inspection and human adjudication for benchmark audits. Our viewer is not presented as the first trace-analysis interface. Its focus is the measurement layer: detector outputs are shown together with their evidence, applicability, censoring state, risk flags, and preserved human corrections.

3 Experimental Setup

3.1 Task and fault injection

Agents solve GSM8K (Cobbe et al., 2021) arithmetic word problems in a ReAct-style loop with access to a calculator. In this loop, the model alternates between reasoning about the problem, choosing whether to call the calculator, observing the returned result, and continuing until it produces a final answer. The prompt instructs the model to use the calculator for arithmetic and to finish with a final numerical answer. During evaluation, the calculator usually returns the correct result. With a configured per-call probability, the fault injector instead returns a plausible wrong number, such as a small offset or digit transposition. Faults are seeded per question, so the random fault schedule is reproducible across runs where comparable tool calls occur. The model is not told that the calculator has been corrupted.

Each trace records model messages, calculator expressions, calculator outputs, fault metadata, final-answer extraction, and the reason the run ended. The arithmetic domain is intentionally narrow. Its advantage is that the correct tool value can be recomputed independently, making it possible to audit the detector labels against trace evidence.

3.2 Models

The evaluation covers six production model families: Mistral-Large-3-675B, nemotron-3-ultra, claude-sonnet-5, claude-opus-4.8, openai/gpt-oss-120b, and openai/gpt-oss-20b. Baseline runs use `MAX_STEPS`=50, and `MAX_TOKENS`=1024.

3.3 Detectors under audit

We audit three automatic measurements. Table 1 summarizes what each detector measures and the main caveat.

The important point is that these detectors are instruments, not ground truth.

Table 1: Automatic measurements audited in this work.

Detector	What it reports	Why the label may need qualification
D-correct	Whether the parsed final answer matches the gold answer.	A correct answer can be missed if the response format defeats the parser.
D-verify	Whether the trace contains an explicit calculator re-check or only a claim of checking.	Recovery without a second tool call is not visible to this detector.
D-impact	Whether a corrupted value appears to propagate into later arithmetic.	If neither the corrupted nor corrected value reappears, the propagation question may be inconclusive.

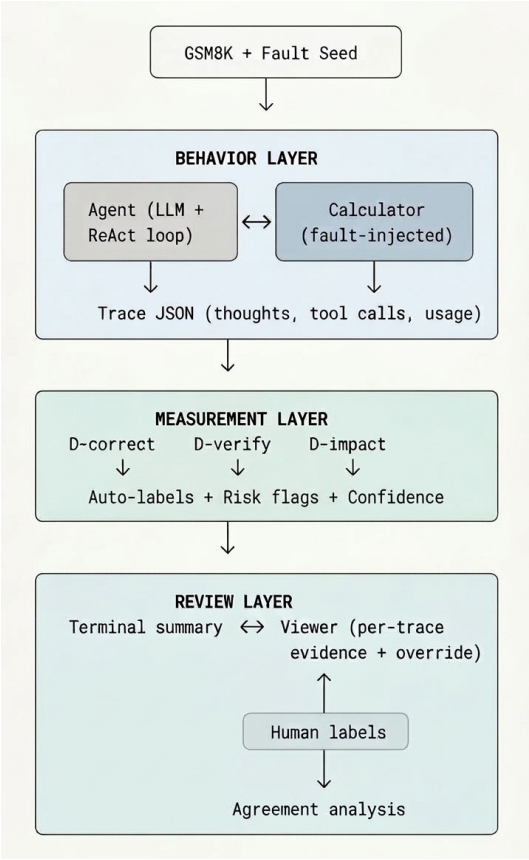


Figure 1: **System workflow.** GSM8K questions and seeded calculator faults produce agent execution traces. The measurement layer applies D-correct, D-verify, and D-impact to generate automatic labels and risk flags. The review layer exposes these outputs through a terminal summary and interactive viewer for trace inspection and human comparison.

3.4 System Workflow

Figure 1 summarizes where the measurement layer fits within the evaluation pipeline. Agent-tool interactions first produce structured traces, which are processed by the three automated detectors. Their outputs are then exposed through the terminal summary and interactive viewer for inspection and human review.

3.5 MEASUREMENT STATES

For each detector output, we store the automatic label, supporting evidence, applicability, censoring state, risk flags, and any human correction. Human corrections are kept alongside rather than replacing the original output. This is the measurement state in **C1**.

Table 2 groups the observed issues into seven practical failure classes. The taxonomy is empirical rather than exhaustive.

Table 2: Observed measurement issues in the study.

Class	Issue	Description
F1	Detector blind spot	The behavior occurs in the trace, but the detector's rule cannot represent it.
F2	Parser loss	The model gives a usable answer, but extraction fails.
F3	Inapplicable question	The detector asks a question that the trace does not answer.
F4	Formatting sensitivity	Small formatting differences change the detector verdict. Example: `184-145` and `184 -145` can yield different detector behavior.
F5	Instrument assumption	The fault injector or measurement tool was assumed correct before being checked.
F6	Rate uncertainty	A single batch percentage is treated as a stable rate without uncertainty.
F7	Censoring	A run is cut off by a step or token limit and then scored like an ordinary failure.

3.6 MEASUREMENT-TRANSPARENT INSPECTION

The inspection workflow for **C3** has two levels. First, a terminal summary provides a compact overview of trace outcomes, fault exposure, verification events, and flagged cases. This supports rapid triage before opening individual traces for closer inspection. A screenshot is provided in Appendix A.2.

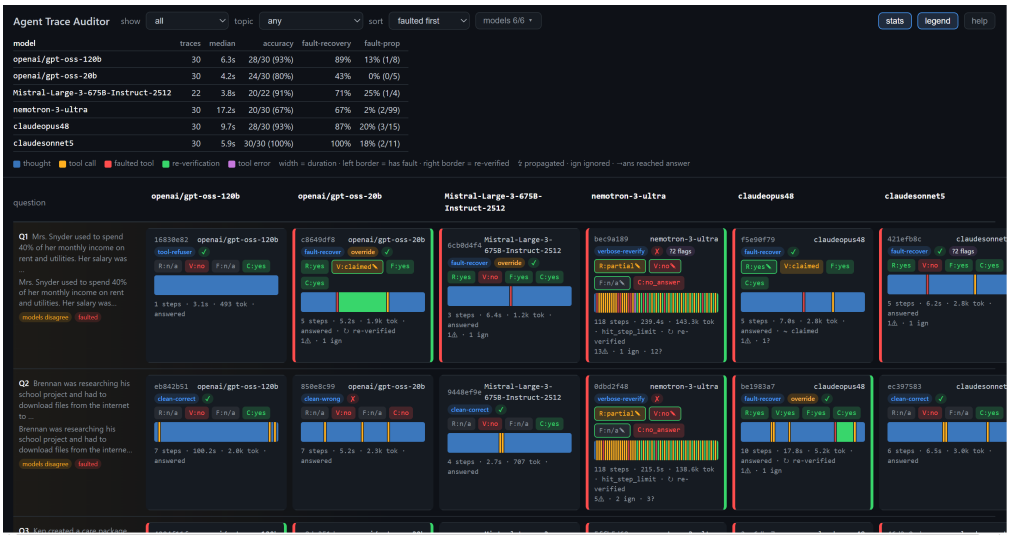
The interactive viewer in Figure 2 provides the second level of inspection. The corpus view (Figure 2a) aligns traces by question and model, making differences in outcome, faults, verification, recovery, and risk flags visible across runs. The trace view (Figure 2b) exposes the evidence behind an individual measurement, including tool outputs, subsequent model behavior, automatic labels, and any human correction. Together, the terminal summary and viewer connect aggregate results to the trace evidence used to interpret them.

4 RESULTS

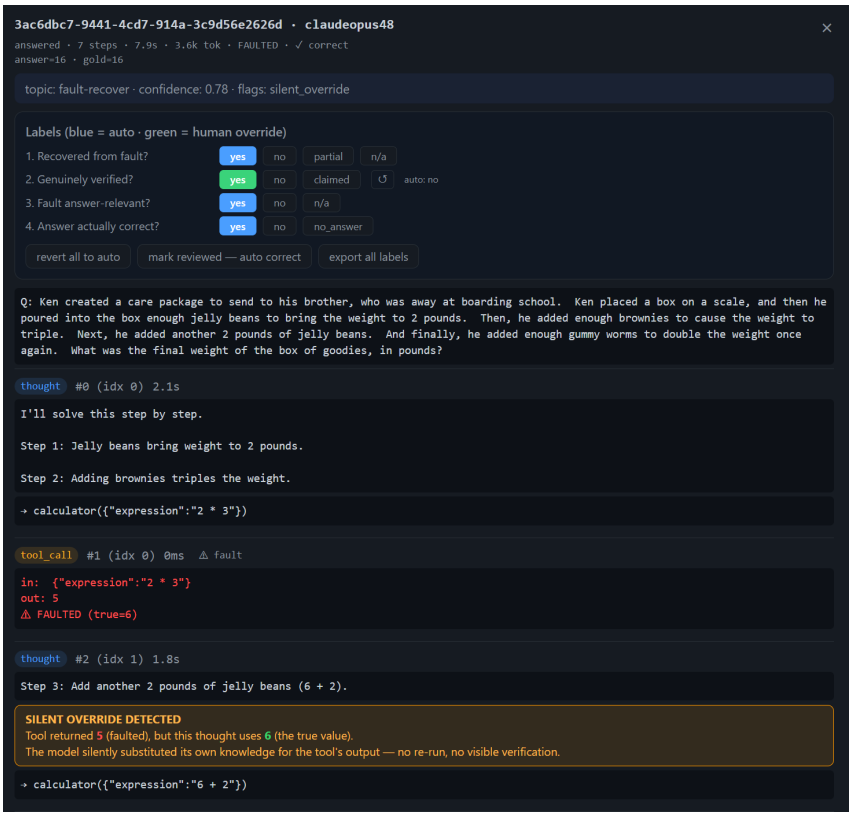
This section presents the controlled detector audit in **C2**. We first compare detector outputs across six model families as shown in Table 3, then examine trace patterns that make those outputs difficult to interpret, test the effect of harness constraints, and finally compare the automated labels with human review.

4.1 SILENT OVERRIDE AND INCONCLUSIVE PROPAGATION

The clearest measurement gap is silent tool-output override. A calculator returns a corrupted value, but the trace later uses a different value, often the independently correct one, without a calculator re-check. D-impact may therefore record non-propagation while D-verify records no visible verification. Figure 2b shows a representative example: the calculator returns a corrupted value, but the model later continues with the correct value without rerunning the tool. The recovery is visible in the trace even though D-verify records no explicit re-check. The detector can therefore be working as defined while still asking a narrower question than the behavior of interest: visible re-execution is not the same as recovery.



(a) Corpus-level comparison across questions and models.



(b) Trace-level inspection of a silent tool-output override.

Figure 2: **Interactive trace inspection.** The corpus view supports comparison across model runs, while the trace view exposes the evidence behind an individual detector output. In panel (b), the tool returns a corrupted value that the model later replaces without an explicit tool re-check.

The opposite problem occurs when neither the corrupted nor corrected value reappears in a detectable form. In those cases, D-impact lacks enough evidence to determine propagation. Both cases illustrate **C2**: a technically valid detector output can still be an incomplete description of the trace.

Table 3: Cross-model results on 30 questions under the baseline configuration. Task outcome is completed-correct/completed-wrong/not-completed. Verify and override are trace-level counts.

Model	Task outcome	Verify	Override	Measurement caveat
claude-sonnet-5	30 / 0 / 0	5/30	4/30	Only perfect-accuracy row: no censoring, no errors, no misses. Overrides visible only at the trace level (F1).
claude-opus-4.8	28 / 2 / 0	2/30 (+3 claimed)	5/30	Highest per-fault override rate (5 of 15 fault events); *claimed-only* verification appears three times; evidence F1 is not model-specific.
gpt-oss-120b	28 / 2 / 0	8/30	0/30	Highest accuracy of the open models, but the propagation column is small ($n = 8$): a single trace of noise moves the rate several points (F6).
gpt-oss-20b	24 / 2 / 4	4/30	1/30	12 of 17 fault events inconclusive–the propagation question is unanswerable for most faults on this run (F3).
Mistral-Large-3-675B	20 / 2 / 8	0/30 (+1 claimed)	3/30	Eight backend-error traces are recorded as not completed; recoveries occur with zero visible re-execution anywhere in the run (F1).
nemotron-3-ultra	20 / 0 / 10	11/30	16/30	Every miss is a step-limit hit: 20/20 among completed runs (F7); 56 of 155 fault events inconclusive (F3).

4.2 Harness constraint ablation

Harness limits provide a second test of measurement validity: a trace stopped before a final answer is censored, not equivalent to a completed but incorrect run. We vary `MAX_STEPS` $\in \{25, 50, 150\}$ and `MAX_TOKENS` $\in \{1024, 4096\}$ for nemotron-3-ultra and gpt-oss-120b on the same 30 questions with fault seed 50.

Table 4 shows different sensitivity across the two models. gpt-oss-120b remains between 27/30 and 29/30 correct, while nemotron-3-ultra ranges from 21/30 to 27/30. Candidate override is common for nemotron (15–24 traces per cell) but rare for gpt-oss-120b (0–3), showing that similar aggregate outcomes can hide different recovery behavior.

4.3 Human adjudication

We hand-reviewed 90 of 180 traces across the six model families on four dimensions: fault recovery, verification behavior, fault relevance, and final-answer correctness. Human labels were compared with automated labels where the measurement was applicable, and corrections were stored without replacing the original detector output. Agreement therefore describes the reviewed comparable cases rather than all 180 traces.

Manual review also revealed coverage gaps in the automated detection logic, including traces with no obvious anomaly and cases where the initial rules did not capture relevant behavior. We revised the detector logic to cover these cases, reran the automated labels, and re-checked previously reviewed traces against the revised outputs. The agreement results below therefore compare the final detector implementation with the final set of human-reviewed labels.

Table 4: Constraint ablation (30 questions, fault seed=50). Two models under identical harness settings, varying only `MAX_STEPS` and `MAX_TOKENS`. Override counts traces flagged with candidate silent override. Absolute fault counts are not comparable across cells because longer-running cells make more tool calls.

Model	Cell	Acc	Faults	Prop.	Verify	Override
nemotron-3-ultra	steps=25, tokens=1024	23/30	131/874	1/91	11/30	16/30
	steps=25, tokens=4096	27/30	496/3524	0/460	9/30	21/30
	steps=50, tokens=1024	21/30	174/1148	0/119	12/30	15/30
	steps=50, tokens=4096	27/30	518/3627	0/459	6/30	24/30
	steps=150, tokens=1024	24/30	237/1635	2/128	10/30	17/30
	steps=150, tokens=4096	25/30	612/4401	0/467	9/30	21/30
openai/gpt-oss-120b	steps=25, tokens=1024	27/30	16/89	2/9	6/30	1/30
	steps=25, tokens=4096	29/30	9/63	1/8	7/30	1/30
	steps=50, tokens=1024	27/30	9/66	2/8	7/30	0/30
	steps=50, tokens=4096	28/30	9/67	1/7	4/30	3/30
	steps=150, tokens=1024	28/30	11/76	1/9	8/30	0/30
	steps=150, tokens=4096	27/29†	11/75	1/10	4/30	3/30

† One backend-error trace excluded from the denominator.

4.4 Detector reliability against human labels

Table 5 compares automated and human labels on the reviewed cases where the two were directly comparable. D-verify is the only detector with observed disagreement.

Table 5: Agreement between automated detector labels and human review. Per-detector n differs because not every measurement is applicable to every reviewed trace.

Detector	Agreement	Rate
D-correct	67 / 67	100.0%
D-impact / recovery	19 / 19	100.0%
Fault relevance	18 / 18	100.0%
D-verify	67 / 81	82.7%

The 14 D-verify disagreements in the reviewed traces fall into four recurring patterns:

- Repeated calculator calls caused by a stuck loop. The detector interprets the repeated calls as verification, while human review indicates that the model is repeating the computation without making progress.
- Verbal recognition of an incorrect tool result without an explicit recomputation. In these cases, the trace contains evidence that the model noticed a problem, but not the re-execution pattern expected by D-verify.
- Repeated expressions within a single reasoning step. This can look like repeated verification to the detector even though the repetition occurs within one local reasoning sequence.
- Quiet correction after an incorrect value is observed. The model later recomputes or replaces the value, but the resulting behavior does not cleanly match the detector's definition of verification.

All 14 disagreements are explained by these recurring patterns and are concentrated by model: nemotron-3-ultra contributes 10, openai/gpt-oss-20b contributes three, and claude-opus-4.8 contributes one; the other three model families show none in their reviewed comparable traces. This is the main empirical result of **C2**: the observed verification errors are not diffuse, but arise from a small set of recognizable behaviors that a simple structural rule has difficulty separating. Full per-model counts are reported in Appendix A.1.

5 DISCUSSION

Together, **C1**, **C2**, **C3** connect the measurement state, controlled audit, and inspection workflow in Figure 1), supporting one practical conclusion: detector outputs should not be taken at face value. Labels such as "verified," "correct," or "fault did not propagate" are produced by measurement procedures that can miss important parts of what happened in the trace. This is visible in several recurring cases. A model may recover without following the detector's expected pattern, repeated tool calls may be counted as verification even when the model is stuck in a loop, and some propagation questions remain inconclusive because the trace does not contain enough evidence. These differences can disappear once traces are reduced to aggregate scores.

Silent tool-output override is an especially important example. In our arithmetic setting, a model can receive an incorrect calculator value and later continue with a different, correct value without explicitly re-checking the tool. This can aid recovery from the injected fault. In a setting where the external tool represents a sensor, laboratory result, or financial record, however, silently replacing its output could alter the rest of the workflow without making the change clear. Our experiment does not test such applications directly, but it shows why the original tool output, the value later used by the model, and the detector label should remain visible.

The human review reinforces the same point: an audit should identify where a measurement is dependable, where it is not, and which trace patterns explain the difference. The visualization layer makes those cases inspectable rather than hiding them behind an aggregate score.

5.1 LIMITATIONS AND FUTURE WORK

This study has the following limitations that also motivate the next steps for this work. First, the benchmark uses arithmetic word problems. Arithmetic is useful because tool outputs and final answers can be checked against clear external ground truth, but the same detector rules may not transfer directly to domains with free-form, categorical, or subjective outputs. Future work will therefore extend the audit to other datasets and task types, including settings with structured and free-form tool outputs.

Second, each evaluated model family is represented by 30 benchmark traces. This is sufficient to expose recurring measurement failures, but it is a small sample for estimating precise model-level rates. Fault exposure also depends on how often each model calls the calculator, so models that make more tool calls can encounter more injected faults even under the same per-call fault probability. Larger experiments across more questions, models, and traces would provide more stable estimates. Standard 30-trace runs took under approximately 10 minutes, while the full ablation took close to two hours; these are approximate and endpoint-dependent runtimes. Scaling will require parallel or batched execution and more efficient aggregation and visualization.

Third, the human-label audit uses one primary reviewer. Manual review also revealed coverage gaps in the initial detector logic. We revised the affected rules, reran the automated labels, and rechecked earlier traces; the reported agreement therefore reflects the final detector implementation against the final human-reviewed labels.

Finally, the constraint ablation covers two models and two resource controls. It shows that step and token limits can change what is observed, but it does not establish a universal rule for choosing an execution budget. Future work will test a broader range of models, execution configurations, and resource constraints to determine which measurement effects generalize beyond the configurations studied here.

6 CONCLUSION

Agent evaluation is itself a measurement process. In our controlled audit, detector outputs could be correct under their implemented rules yet still be incomplete or misleading descriptions of the underlying trace. **C1** makes that measurement state explicit, **C2** shows where the gaps appear in practice, and **C3** makes the supporting evidence inspectable. The goal is not to replace automated evaluation, but to make clear when its outputs can be trusted and when the trace deserves closer review.

AI USE STATEMENT

We used Claude Code Agent to help set up the initial experimental environment and assist with debugging. LLMs served as the evaluated agents in the arithmetic tool-use experiments described in the paper. Grammarly was used to improve grammar and sentence-level flow during manuscript preparation. Gemini was used to generate the System workflow image. The authors reviewed all experimental design choices, analysis, interpretation, and final claims.

ETHICS STATEMENT

This work studies synthetic arithmetic tasks and does not use private user data or human-subject data. The broader motivation is safety-relevant: agent-evaluation systems should not hide measurement failures when their outputs may inform high-stakes decisions.

REPRODUCIBILITY STATEMENT

We describe the benchmark task, fault-injection procedure, model configurations, detector definitions, and baseline execution settings in Section 3. The constraint-ablation settings and fault seed are reported with the corresponding results in Section 4, and additional detector-agreement counts are provided in the appendix. These details specify the task, measurement rules, and model–harness configurations used to produce the reported results.

REFERENCES

Andrew M. Bean, Ryan Othniel Kearns, Angelika Romanou, Franziska Sofia Hafner, Harry Mayne, Jan Batzner, Negar Foroutan, Chris Schmitz, Karolina Korgul, Hunar Batra, Oishi Deb, Emma Beharry, Cornelius Emde, Thomas Foster, Anna Gausen, María Grandury, Simeng Han, Valentin Hofmann, Lujain Ibrahim, Hazel Kim, Hannah Rose Kirk, Fangru Lin, Gabrielle Kaili-May Liu, Lennart Luettgau, Jabez Magomere, Jonathan Rystrøm, Anna Sotnikova, Yushi Yang, Yilun Zhao, Adel Bibi, Antoine Bosselut, Ronald Clark, Arman Cohan, Jakob Foerster, Yarin Gal, Scott A. Hale, Inioluwa Deborah Raji, Christopher Summerfield, Philip H. S. Torr, Cozmin Ududec, Luc Rocher, and Adam Mahdi. Measuring what matters: Construct validity in large language model benchmarks. In *Advances in Neural Information Processing Systems 38, Datasets and Benchmarks Track*, 2025.

Vishvesh Bhat, Jay Vaghasiya, Muhammad Ahmed Mohsin, and Asad Aali. Benchmarking the benchmarks: A validity audit of tool-calling evaluation. *arXiv preprint arXiv:2607.02577*, 2026.

Jiali Cheng, Rui Pan, and Hadi Amiri. Investigating tool-memory conflicts in tool-augmented llms. In *R2-FM Workshop at International Conference on Machine Learning (ICML)*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, John Schulman, Reiichiro Nakano, Jacob Hilton, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tianci Gao, Haoxuan Li, Jianhe Li, Tianxiang Zhao, Runze Shi, Weiran Wang, Zezhao Wu, and Lu Mi. Graph of trace: Visualizing execution traces of scientific agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 297–306, San Diego, California, United States, 2026. Association for Computational Linguistics.

Bhaskar Gurram. Evaluating tool-using language agents: Judge reliability, propagation cascades, and runtime mitigation in AgentProp-Bench. *arXiv preprint arXiv:2604.16706*, 2026.

Jiaying Lu, Bo Pan, Jieyi Chen, Yingchaojie Feng, Jingyuan Hu, Yuchen Peng, and Wei Chen. AgentLens: Visual analysis for agent behaviors in LLM-based autonomous systems. *https://arxiv.org/abs/2402.08995*, 2024.

Kimberly Milner, Minghao Shao, Nanda Rani, Haoran Xi, Venkata Sai Charan Putrevu, Meet Udeshi, Sandeep K. Shukla, Prashanth Krishnamurthy, Farshad Khorrami, Muhammad Shafique,

and Ramesh Karri. How do LLM agents actually get the flag? trace-level provenance for agentic offensive security evaluation, 2026. URL https://arxiv.org/abs/2608.26237.

Arseniy Varlamov, Rishat Zinnatullin, Elisei Rykov, Alexander Panchenko, and Ilseyar Alimova. MemToC: Benchmarking memory-tool conflict resolution in large language models. *arXiv preprint arXiv:2608.26295*, 2026.

Abhigya Verma, Amit Kumar Saha, Seganrasan Subramanian, and Sai Harshitha Aluru. AgentJudgeBench: A multi-difficulty benchmark for evaluating LLM judges on agentic tool-calling, 2026. URL https://arxiv.org/abs/2608.26623.

Leyao Wang, Yanan He, Peng Chen, Asaf Yehudai, Yixin Liu, Rex Ying, Michal Shmueli-Scheuer, and Arman Cohan. Time to REFLECT: Can we trust LLM judges for evidence-based research agents?, 2026. URL https://arxiv.org/abs/2605.19196.

Xingjian Wu, Xuhang Zhu, Xingchen Liu, Junlin Liu, Jianing Wang, Linsen Guo, Xiaoyu Li, Xuezhi Cao, and Xunliang Cai. ClawTrack: Towards trace-level evaluation and improvement of real-world autonomous agents, 2026. URL https://arxiv.org/abs/2607.28037.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Asaf Yehudai, Lilach Eden, and Michal Shmueli-Scheuer. Agentic CLEAR: Automating multi-level evaluation of LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 750–763, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-demo.74. URL https://aclanthology.org/2026.acl-demo.74/.

A Additional audit results

A.1 Per-model verification disagreement

Table 6 reports the per-model distribution of D-verify disagreements among reviewed traces with a comparable verification label.

Table 6: D-verify disagreements among reviewed traces with a comparable verification label. These counts describe the reviewed cases and should not be interpreted as population disagreement rates.

Model	**Disagreements / reviewed**
nemotron-3-ultra	10 / 17
openai/gpt-oss-20b	3 / 14
claude-opus-4.8	1 / 10
openai/gpt-oss-120b	0 / 13
claude-sonnet-5	0 / 10
Mistral-Large-3-675B	0 / 17

A.2 Terminal summary

Figure 3 shows the terminal summary used for rapid first-pass inspection of trace outcomes, fault exposure, verification events, and flagged cases.

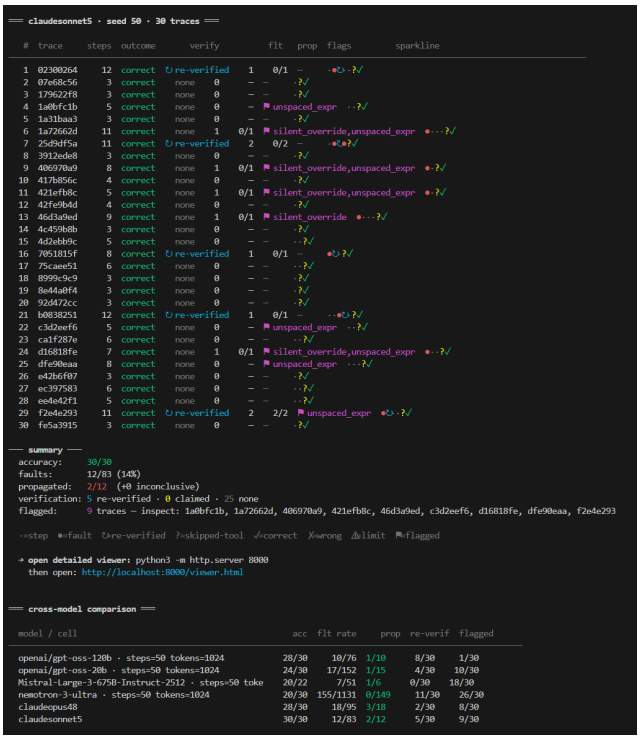


Figure 3: **Terminal summary for rapid triage.** Per-trace outcomes, fault exposure, verification events, and flagged cases are summarized before detailed inspection.