RITE: EVALUATING END-TO-END RESILIENCE TO SILENT TOOL ERRORS IN E-COMMERCE AGENTIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using LLM agents increasingly answer e-commerce queries through multi-step tool interactions. The expected output of a tool call can shift between agent development and execution, allowing a plausible but incorrect value to be returned without an explicit failure signal. These silent errors may propagate through subsequent decisions into final answers. To study agents' resilience across this end-to-end process, we formulate Silent-error Resilience Task, an interactive task that evaluates whether they complete the original query after a silent error corrupts an intermediate state. We introduce RITE, a benchmark built from human-authored industrial workflows and production-log-derived inputs. To enable controlled and reproducible evaluation, we use a semantic error taxonomy to inject errors into fixed workflow prefixes, freeze recorded tool behavior, and let agents continue freely. It contains 167 matched units across nine e-commerce workflows, yielding 501 silent-error, correct-value, and explicit-failure instances. Across 10 recent LLMs, silent errors roughly halve task success and are consistently harder than explicit failures. Recognition, use of an available verification path, and integration of the resulting evidence form successive bottlenecks. We also propose a light State Checker as an intervention to test whether resilience can be improved.

1 INTRODUCTION

Agentic LLM systems increasingly complete e-commerce tasks through multi-step workflows that invoke multiple external tools and pass their outputs from one step to the next. In industrial deployments, a tool's runtime output may differ from what the agent was built to expect because underlying business data, tool implementations, or deployed versions can change between development and execution. This build-time–runtime mismatch can remain hidden: a call may complete and conform to the expected API schema while returning a plausible value that is incorrect for the current task. We refer to such plausible-but-wrong intermediate values without an explicit failure signal as *silent errors*. For example, suppose a user mainly watches local football videos, one of which mentions blockchain-based fan tokens. During agent development, the profile tool returns "local sports"; after an update that overweights "blockchain," it returns "blockchain" at execution. The new label remains plausible and schema-valid, so no failure is signaled. The absence of an explicit failure signal makes silent errors particularly severe: an agent may have no immediate basis for detecting that the value is incorrect. It may therefore propagate the corrupted value through subsequent tool calls, allowing a single intermediate error to directly alter the final user-facing output.

Existing benchmarks evaluate tool use in nominal settings (Qin et al., 2024; Trivedi et al., 2024; Ma et al., 2024), recovery from explicit or implicit faults injected while agents run (Zhu et al., 2026; Tian et al., 2026; Chen et al., 2026), silent-error detection (Sun et al., 2024; Mazumder & jahan Lia, 2026), or how corrupted intermediate values affect later outputs (Gurram, 2026). In these fault-based settings, each agent runs freely and meets a fault only if and when its own trajectory reaches the perturbed call, so different agents face silent errors at different points and with different histories. They therefore cannot compare different agents on the same silent error at the same decision point, nor pair that error with correct-value and explicit-failure counterparts at that point.

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.

To address this gap, we formulate Silent-error Resilience Task, in which a plausible but incorrect tool result enters an agent's execution history without an explicit failure signal. Given the query and toolbox, the agent operates in an agentic loop, repeatedly reasoning, calling tools that may return silent errors, and incorporating their outputs until it produces a final answer. The task asks whether the agent can recognize the error, reacquire reliable evidence, and complete the original query instead of propagating the error into its final answer.

To evaluate this task under controlled conditions, we introduce RITE (Resilience to Incorrect Tool Evidence). Each human-authored workflow specifies a pre-orchestrated agentic process, and its execution log records tool calls and observations in the same format as an agent trajectory. We use this log as a fixed trajectory, allowing us to control where a silent error is introduced. We select a semantically important slot in the log and inject a silent error at that point. The trajectory up to and including this slot is provided as the agent's prefix, while the workflow's tools and their recorded input–output behavior are frozen to form the evaluation world, so that recorded calls return identical outputs across models and conditions. The agent then continues in an agentic loop, allowing us to compare its behavior while controlling for error type, slot, and environment.

Following this protocol, we first construct a semantic taxonomy covering errors in object or category selection and in attributes or values. We sample inputs from the production logs of one internal platform and use them to run workflows deployed on a separate internal platform. From the resulting traces, we select task-relevant checkpoints and instantiate taxonomy-guided plausible but incorrect values. Each resulting unit contains matched prefixes for the silent-error, correct-value, and explicit-failure conditions, all paired with the same evaluation world. RITE contains 167 matched units from nine e-commerce workflows, yielding 501 instances.

We evaluate 10 recent closed and open-weight LLMs on RITE. Across all models, silent errors reduce task success to roughly half of correct-condition performance and are consistently harder to handle than matched explicit failures. Our analysis traces this weakness through a three-stage chain: agents often fail to *recognize* that an inherited value may be wrong; recognition does not reliably lead them to *refetch* its source; and refetching does not guarantee that the corrected value is carried through to final *recovery*. We further find that most benchmark corruptions would change the downstream e-commerce task output, with damage varying substantially by error type and workflow position. Finally, a lightweight State Checker makes agents notice and refetch more often but yields only limited improvement in final task success. Making agents look is therefore easier than making them repair what they find. In summary, our contributions are:

- **Task formulation.** We formulate Silent-error Resilience Task as an interactive agentic task for studying resilience to silent errors throughout the query-to-answer process. It evaluates whether an agent can recognize an unreliable tool result, reacquire trustworthy evidence, and still produce a correct final answer rather than propagate the error downstream.
- **Benchmark construction.** We introduce RITE, which replays a human-authored workflow execution up to a selected point, returns a correct, silently incorrect, or explicitly failed result there, and lets the agent continue freely. Using inputs sampled from one internal platform's production logs, workflows deployed on another, and a semantic error taxonomy, we construct 167 matched units across nine e-commerce workflows, comprising 501 silent-error, correct-value, and explicit-failure instances.
- **Empirical analysis.** Across 10 LLMs, silent errors roughly halve task success. We identify recognition, producer refetch, and recovery as successive failure points, characterize effects by error type and position, and propose a lightweight State Checker as an intervention to test whether resilience can be improved.

2 Related Work

Tool-use and agent evaluation. Agent evaluation spans three levels. At the tool-call level, MetaTool (Huang et al., 2024b) asks whether a tool is needed and which one to select, while ToolBench, BFCL, and API-Bank (Qin et al., 2024; Patil et al., 2025; Li et al., 2023) evaluate call construction and extend to multi-tool, dependent, and stateful multi-turn interactions; ToolACE (Liu et al., 2025) provides synthesized data for improving these abilities. At the task-outcome level, AppWorld, GAIA, and WebArena (Trivedi et al., 2024; Mialon et al., 2024; Zhou et al., 2024) judge a complete

rollout by its final answer or by the state it leaves in the environment, while allowing different valid action paths. At the process level, AgentBoard (Ma et al., 2024) tracks task progress, WorFBench (Qiao et al., 2025) evaluates generated workflow structure, ToolScan (Kokane et al., 2025) diagnoses call-level errors, and process-mining methods (van der Aalst, 2016) compare executed and expected flows. Across these levels, the benchmarks above vary task difficulty and agent behavior but do not systematically manipulate the semantic correctness of intermediate tool values. We instead vary the intermediate business value and ask whether an agent can still complete the task correctly when that value is wrong. To keep the test controlled and reproducible, we provide the recorded workflow execution up to the selected checkpoint as the agent's initial history, then let it continue freely.

Tool-failure recovery, silent-error detection, and error propagation. Work on unreliable tools can be separated by the stage of the error-handling chain that it evaluates. One line studies recovery from tool failures: agents resume from the environment left by a failed attempt (Tan et al., 2025), rewind to an earlier checkpoint (Zhuang et al., 2026), recover from explicit or implicit faults injected into tool responses while the agent runs (Zhu et al., 2026; Tian et al., 2026; Chen et al., 2026), or diagnose failed trajectories and characterize recovery cost (Vuddanti & Chittiprolu, 2026; Zhu et al., 2025; Zhang et al., 2026). Resumption settings start from uncontrolled natural failures, while injection settings expose an agent to a fault only if and when its own trajectory reaches the perturbed call. A second line studies silent-error detection, asking whether agents identify faulty outputs that trigger no exception (Sun et al., 2024) or silent data-quality faults introduced during controlled replay (Mazumder & jahan Lia, 2026). In both, the faulty response is delivered when the agent's own run issues the corresponding call. A third line studies error propagation, tracing a schema-valid but semantically wrong argument to the final answer (Gurram, 2026) and measuring how perturbed requests or tool responses alter later calls and task outcomes (Xiong et al., 2025; Wang et al., 2026b). To our knowledge, RITE is the first benchmark to evaluate business-semantic silent errors at controlled decision points: every agent encounters the same plausible but incorrect value after the same recorded execution history, the same point is also instantiated with the correct value and with an explicit failure, and the agent then continues freely to a final answer.

E-commerce and business-domain agents. Work in this domain spans three forms of capability. Domain models test whether a model understands products and users well enough to answer questions (Peng et al., 2024), while recommender agents use that understanding to retrieve, rank, and recommend products (Gao et al., 2023; Huang et al., 2024a). Shopping benchmarks place agents in interactive stores, where they must search, compare, and purchase while satisfying constraints on products, vouchers, budgets, and sellers (Yao et al., 2023; Wang et al., 2026a). Business-service benchmarks require agents to interact with users over multiple turns, operate on business records through tools, and comply with domain policies (Yao et al., 2025; Wang et al., 2025), while CRMArena-Pro additionally tests confidentiality in enterprise workflows (Huang et al., 2026). These benchmarks do not systematically test what happens when a task-critical business value is silently corrupted. Grounded in production execution logs and human-authored industrial e-commerce workflows, our benchmark instead examines this reliability risk involving tools developed and maintained by multiple parties: a call may complete while an intermediate business value is silently wrong.

3 Problem Formulation

We study whether an agent can remain reliable when a tool silently returns an incorrect result and still complete the original task. We first formulate this end-to-end resilience as Silent-error Resilience Task, then define how RITE evaluates it in this section.

Silent-error Resilience Task. In a standard agentic task, an agent A receives a task query q and repeatedly selects and executes tool calls from a tool set $\mathcal{T}$ provided by an environment E. The interaction begins without a preloaded execution history, i.e., $h_0^{\mathrm{std}} = \varnothing$. At each step t, the agent observes the current history h_t and selects an action a_t. The environment executes the action and returns an observation o_t, which is appended to the history:

$$a_t = A(q, h_t, \mathcal{T}), \qquad o_t = E(a_t), \qquad h_{t+1} = h_t \oplus (a_t, o_t). \tag{1}$$

The standard action set is $\mathcal{A}_0 = \{\mathrm{call}, \mathrm{reason}, \mathrm{stop}\}$. The interaction ends when the agent stops or reaches the step budget, producing a final answer.

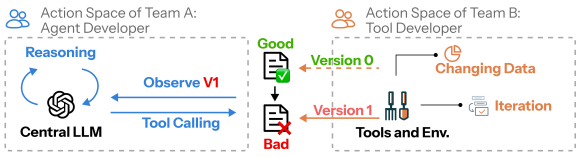


Figure 1: Silent errors from build-time–runtime mismatch. Tool-side data or implementation changes can make a runtime result diverge from what the agent was built to expect while signaling no failure.

As illustrated in Figure 1, the standard loop does not guarantee that a returned observation is semantically correct. In deployed systems, changes to business data, tool implementations, or deployed versions can cause runtime results to diverge from the expectations under which the agent was built. A call may still complete and conform to its API contract while returning a result that is incorrect for q. At a tool-call step j, let $o_j^\star$ denote the task-correct result and $\tilde{o}_j$ the result actually returned by the tool. A *silent error* occurs when $\tilde{o}_j$ differs from $o_j^\star$ in task-relevant meaning but remains plausible, schema-valid, and free of an explicit failure signal. Because the ordinary update rule still appends $(a_j, \tilde{o}_j)$ to the history, subsequent actions condition on unreliable evidence and may propagate it into the final answer.

The Silent-error Resilience Task measures whether the agent can still complete q after such a result has entered its execution history. Process measures track whether the agent recognizes the unreliable result, reacquires trustworthy evidence, and integrates that evidence into a correct final answer. The agent retains $\mathcal{A}_0$ and may reason from the current state, revisit or reissue an earlier tool call, seek corroborating evidence, or pursue another continuation.

Benchmark evaluation. For each benchmark unit i, RITE constructs three matched instances $D_i^k = (q_i, p_i^k, c_i)$ for $k \in \{\text{correct}, \text{implicit}, \text{explicit}\}$. The instances share the query q_i, result constraint c_i, and continuation environment E_i, keeping the user task, required result, and available recovery paths fixed. They differ only in the affected evidence in p_i^k: the correct condition provides the recorded value as a baseline, the implicit condition provides a silently incorrect value, and the explicit condition provides an overt failure to expose the effect of a visible failure signal.

For each condition, RITE initializes the agent with $h_{0,i}^k = p_i^k$ and provides q_i as the task query. Starting from the matched checkpoint controls variation in the preceding execution, while free continuation in E_i with the standard action set $\mathcal{A}_0$ lets the agent choose its own recovery path. The run ends when the agent stops or reaches the step budget. Let $\hat{r}_i^k$ denote the agent-generated continuation, which ends with a final answer y_i^k; the complete trace is $r_i^k = p_i^k \oplus \hat{r}_i^k$.

We use three evaluation measures. *Noticed* records whether the agent explicitly identifies a content or result anomaly and takes a related remedial action. Result Success Rate (RSR) records whether y_i^k resolves q_i, capturing end-to-end task success. Retention, the ratio of implicit-condition RSR to correct-condition RSR, reports how much baseline performance is preserved under silent errors.

4 Benchmark Construction and Evaluation Environment

Each human-authored workflow specifies a pre-orchestrated agentic process. Its execution produces a controlled trace of tool calls and observations in the same format as an agent rollout. We use this trace to construct matched prefixes that share the preceding history and differ only in selected intermediate evidence; all prefixes are paired with the same continuation environment. This section first describes the data sources and silent-error taxonomy, then explains how each clean trace yields paired benchmark instances D_i^k and an evaluation environment E_i, and reports its distribution.

4.1 Data Sources and Silent-Error Taxonomy

Data sources. Benchmark construction begins with two source collections. The first is the workflow collection $\mathcal{W}$, drawn from human-authored e-commerce workflows already deployed and running on an internal industrial workflow platform. We screen these existing workflows for structural validity and task diversity, yielding nine golden workflows with explicit control flow and deterministic tool dependencies. The second source is the input collection $\mathcal{X}$ from a separate internal e-commerce data platform. Candidate production-log records are mapped to workflow inputs and selected only when their executions yield complete traces with task-relevant intervention points; the corresponding traces are retained. Appendix A defines the workflow- and log-selection criteria.

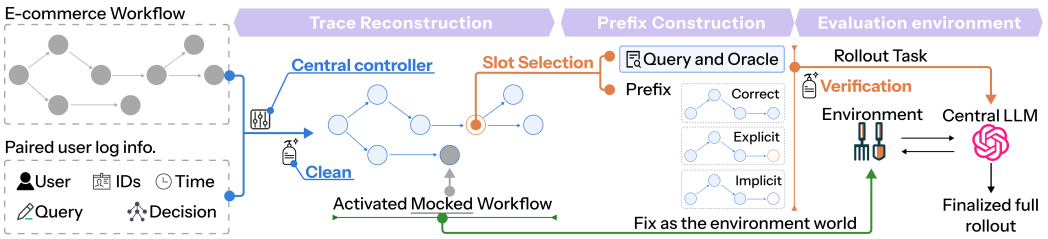


Figure 2: RITE benchmark construction and rollout pipeline. A workflow W_i and paired log-derived input x_i are used to activate a side-effect-free workflow replica and reconstruct a clean trace τ_i. A task-relevant slot then yields a shared query and result oracle together with three matched prefixes (correct, explicit, and implicit). The workflow and clean trace define the fixed evaluation environment E_i, in which the evaluated LLM continues from each prefix to produce a complete rollout.

Taxonomy definition. We constructed the silent-error taxonomy through a targeted, literature-grounded synthesis of adjacent error and failure taxonomies. We selected studies whose source categories could clarify the boundary of a silent state corruption or be operationalized as schema-valid, plausible-but-wrong workflow values. For each relevant source category, two authors independently assessed its observable error signal, workflow locus, and suitability for conversion into a single-slot implicit perturbation. They then reconciled overlapping categories and clarified the resulting boundaries. The final taxonomy contains seven leaf types organized into two semantic groups. Object or category selection errors cover substitution with a same-level category (category sibling drift), an ancestor or descendant (category granularity mismatch), or a similar but incorrect real entity (entity near-duplicate). Attribute or value errors cover changes to an entity's attribute (profile attribute drift), movement to a neighboring valid numeric range (numeric bucket shift), superficially similar text with a different business meaning (semantic synonym drift), and reuse of a formerly valid but outdated value (temporal staleness). The benchmark instantiates every leaf type except temporal staleness. Although a stale value can be schema-valid and lead to an incorrect answer, recognizing that it is stale requires prior or time-indexed evidence that the agent does not observe. Appendix B provides the complete definitions, category boundaries, and literature grounding.

4.2 Benchmark Infrastructure Construction

The infrastructure has a shared start followed by two sibling construction branches:

$$(W_i, x_i) \longmapsto \tau_i, \qquad \tau_i \longmapsto D_i^k = (q_i, p_i^k, c_i), \qquad (W_i, \tau_i) \longmapsto E_i.$$

The data branch constructs the paired prefixes initializing an agent, whereas the environment branch independently constructs the world where the agent continues acting (Fig. 2).

Workflow trace reconstruction. To avoid polluting the internal platform data, which may record the workflow execution result, we first post-process every selected workflow into a deployable replica by mocking out reporting and other side-effecting nodes and deploy the modified workflow back onto the platform. Then, a local central controller connects the log and workflow to provide an execution trace. It initially maps the production log record to the start schema of the deployed replica, producing the input x_i. It then invokes the deployed workflow through the platform API with x_i and retrieves the full execution trace of that run, which records every executed node with its arguments, outputs, status, and data dependencies. We retain only executions that complete successfully and contain full information needed to reconstruct the workflow state. The retained τ_i is therefore the clean execution record shared by the two downstream branches.

Paired prefix construction. **Slot selection.** We align the workflow graph with τ_i and obtain an execution graph whose edges carry the values propagated between nodes. We then search this graph along producer $\rightarrow s_i \rightarrow$ consumer $\rightarrow$ downstream result: an intermediate slot s_i qualifies when it carries business-relevant content, is consumed downstream, and plausibly affects the task outcome. For each retained slot, we record its producer and consumer and determine the locus at which the intervention is realized, either in the producer observation (`tool_output`) or in a downstream tool argument (`tool_input`). A single execution yields multiple such slots, and each retained slot defines one task through the prefix built around it.

Task construction. This stage converts each retained slot into a task including the task query and corresponding three paired prefixes.

(i) **Three conditions at the same slot.** We truncate the common trace at the same slot and render the three prefixes p_i^{correct}, p_i^{explicit}, and p_i^{implicit} (Sec. 3), with neutral historical rationales and no corruption marker. The correct condition preserves the recorded value. The explicit condition replaces access to that value with an overt failure, such as an empty argument or an error-bearing observation. The implicit condition obtains its value in two steps. First, it selects an applicable error type from the taxonomy and constructs a substitute that is type- and schema-compatible, remains locally plausible, and changes the task-relevant meaning of the slot. Depending on the error type and carrier, candidates come from compatible values observed in other cases or from a type-specific controlled transformation. Second, the selected substitute is written into the slot at its predetermined locus: the producer observation under `tool_output`, or a downstream tool argument under `tool_input`.

(ii) **Query and oracle.** From the clean trace, we extract the correct value of s_i and construct the result constraint c_i that a correct final answer must satisfy. We then generate or refine q_i so that it is designed to depend on this value: answers grounded in the gold and polluted values should be distinguishable under c_i.

Verification. Before delivery, each candidate must preserve prefix executability and structural compatibility. Four additional gates then check the intervention itself. **(1) Plausibility.** The implicit value must remain schema-valid and locally reasonable while differing from the recorded value in task-relevant meaning. **(2) Explicit-failure clarity.** The explicit condition must expose an unambiguous failure signal. **(3) Non-leakage.** Neither the query nor the prefix may reveal that an intervention was applied. **(4) Discriminability.** The query remains solvable, and answers grounded in the gold value and in the polluted value must remain distinguishable under c_i. The surviving artifacts are delivered in the formulation's schema $D_i^k = (q_i, p_i^k, c_i)$, with all three conditions sharing the same query and result constraint. Providing every model with the same recorded execution before the affected value deliberately reduces variation in reaching that point, so the matched conditions focus on how agents handle the evidence they receive there.

Evaluation environment construction. In parallel with prefix construction, we extract the callable tool set $\mathcal{T}_i$ from W_i and build one replay world R_i for that workflow by integrating the recorded responses of all collected traces.

$$(W_i, \tau_i) \longmapsto E_i(a_t) = \begin{cases} R_i(a_t), & a_t \in \mathrm{dom}(R_i), \\ \mathrm{Live}_{\mathcal{T}_i}(a_t), & \text{otherwise.} \end{cases} \tag{2}$$

Under this interface, a covered call returns the recorded response from R_i, while an unseen valid call is dispatched to the live implementation in $\mathcal{T}_i$ deployed on the internal industrial workflow platform. RITE changes only the condition-specific prefix, not E_i; because the replay world is built from the clean trace, re-querying a recorded producer returns the corresponding unpolluted response, giving the agent a direct path to recover reliable evidence and produce a correct final answer. Every instance is therefore recoverable by design, isolating whether agents use an available verification path; faults that persist across re-queries are discussed in Appendix G.

4.3 Dataset Statistics

The final RITE release contains 167 paired units, each defined by a unique slot–locus pair and expanded into correct, explicit, and implicit conditions. This yields 501 instances, with 167 instances per condition and complete triplets throughout. Nine workflows contribute data, and the release covers six of the seven taxonomy error types because temporal staleness has no retained instance. Of the 167 units, 147 intervene at `tool_output`; the remaining 20 intervene at `tool_input`. Detailed distributions are reported in Appendix F.

5 Experiments

We evaluate 10 recent LLMs in three groups: two closed frontier models (Claude-Opus-4.8 and GPT-5.5), five open-weight frontier models (GLM-5.2, Kimi-K3, DeepSeek-V4-Pro, Hy3,

and Qwen3.5-397B), and three smaller open-weight models (Qwen3.6-35B, Qwen3.5-35B, and Qwen3.5-122B). Each model runs all 501 instances at temperature 0 with a 25-step budget. Rollouts invalidated by environment-side tool anomalies are excluded. The evaluation environment follows Section 4.2, and the rollout and judging protocol is detailed in Appendix C, and a human validation of the Result and Noticed labels is reported in Appendix D. Following Section 3, we report condition-specific RSR and Noticed, Retention, and the paired gap: the paired explicit-minus-implicit RSR difference over slots observed in both conditions.

5.1 Main Results

Table 1 shows that implicit errors substantially degrade task performance across all evaluated models and, in most cases, remain unnoticed. Appendix E provides two end-to-end cases.

Table 1: Results across the three matched conditions, grouped by model class and ordered within each group by implicit-condition Noticed. The Noticed gap is explicit minus implicit. **Bold** marks the best value in each column.

		Noticed			RSR			Resilience	
Category	Model	Implicit ↑	Explicit ↑	Gap	Correct ↑	Implicit ↑	Explicit ↑	Retention ↑	Paired gap
Closed frontier	Claude-Opus-4.8	**0.336**	0.579	0.243	0.765	**0.403**	0.507	**0.526**	**+0.089**
	GPT-5.5	0.280	**0.594**	0.314	0.716	0.363	0.526	0.507	+0.185
Open-weight frontier	Kimi-K3	0.263	0.590	0.327	0.784	0.302	0.555	0.385	+0.228
	GLM-5.2	0.217	0.487	0.270	0.805	0.277	0.507	0.345	+0.243
	Hy3	0.215	0.494	0.278	0.806	0.310	0.552	0.385	+0.236
	DeepSeek-V4-Pro	0.212	0.468	0.255	**0.827**	0.327	0.533	0.395	+0.217
	Qwen3.5-397B	0.210	0.477	0.267	0.809	0.320	0.543	0.396	+0.220
Smaller open-weight	Qwen3.6-35B	0.248	0.512	0.264	0.823	0.335	0.531	0.408	+0.210
	Qwen3.5-35B	0.211	0.512	0.301	0.791	0.340	0.534	0.429	+0.208
	Qwen3.5-122B	0.208	0.506	0.299	0.784	0.302	**0.564**	0.385	+0.265

F1: Silent errors substantially degrade task performance across models. Across all ten models, implicit errors reduce RSR from 0.716–0.827 in the correct condition to 0.277–0.403, yielding Retention of only 0.345–0.526 (Table 1). This loss is tied to the absence of an observable failure signal: when the corresponding failure is explicit, RSR rises to 0.507–0.564, producing a positive explicit-minus-implicit paired gap for every model (+0.089 to +0.265; +0.214 pooled). Stronger task performance does not eliminate this vulnerability. Correct-condition RSR spans only 0.111 across models, whereas Retention spans 0.182; closed frontier models have higher mean Retention than open-weight models (0.517 versus 0.391), yet even the best model preserves barely half of its correct-condition performance. Thus, susceptibility to silent errors is widespread rather than confined to models with weak baseline task ability.

F2: Models rarely notice silent errors or refetch their producers. Across models, implicit-condition Noticed ranges only from 0.208 to 0.336 (Table 1), so most silent errors go unrecognized. Table 2 shows that recognition helps but is not sufficient: the latest producer observation matches the gold value in 0.3% of Unnoticed rollouts and 23.3% of Noticed rollouts, while 57.8% of Noticed rollouts still match the polluted value. To examine whether recognition leads to active verification, we track *producer refetch*, defined as re-invoking the tool that produced the affected value. Among scored Noticed rollouts, RSR is 0.387 without a producer refetch and 0.655 with one. GLM-5.2 illustrates this gap: its Noticed rate is 0.217, but its producer-refetch rate is only 0.071 and its Retention is the lowest at 0.345. Across the eight open-weight models, producer-refetch rate is positively but not significantly associated with Retention ($r = 0.491$, $p = 0.217$). These results identify two bottlenecks: models rarely notice silent errors, and noticing one often does not lead to refetching its source.

Table 2: Latest valid producer observation by Noticed in the implicit condition.

Noticed	n	Gold	Polluted	Other
No	1198	0.3%	99.6%	0.1%
Yes	377	23.3%	57.8%	18.8%

F3: Producer refetch does not ensure recovery. To test whether producer refetch restores the correct evidence, the state judge compares the latest valid producer observation with the gold and polluted values. Among the 147 implicit-condition rollouts with a producer refetch, 87 (59.2%) match the gold value and 60 (40.8%) match neither value; none match the polluted value. Of the 87

matches, 85 are obtained in one re-query and the remaining two require multiple rounds. Among Noticed rollouts with a producer refetch, about one third still fail the task (RSR 0.655; F2).

To understand why refetching does not ensure recovery, we manually inspect the 60 trajectories that did not match value, focusing on unsolved rollouts in which the re-query nonetheless returns usable evidence. After usable evidence is acquired, we identify three recurring patterns. First, *reasoning overwrite* occurs when a model initially uses recovered evidence but later replaces it with a self-imposed heuristic. In one trajectory, an agent reconstructs the correct timestamps and uses them in the next RPC call, but then redefines the "opening warm-up" boundary and selects the wrong final comment. Second, *lossy selection* occurs when a model alters recovered candidates while choosing among them. One agent merges two recovered topic clusters despite an instruction to select one, while another replaces the retrieved topic with a plausible but unsupported reformulation. Third, *unsupported synthesis* occurs when a model uses corrected evidence but adds claims absent from it. Agents incorporate corrected profile or interest labels but add unsupported viewing-duration, interaction, or presentation claims. Across these patterns, usable evidence is obtained, but the correction does not carry through to the final answer.

6 Discussion

The main results separate silent-error handling into recognition, use of an available verification path, and integration of the resulting evidence into final recovery. This section asks three follow-up questions: whether the resulting performance loss corresponds to changed e-commerce task outputs (D1), where the vulnerability concentrates (D2), and how far a light check can go (D3).

D1: Do silent errors change e-commerce task outputs? Not every corrupted task-relevant value changes the downstream e-commerce output. We therefore use an LLM judge to identify slots where answering with the gold versus the polluted value would change the business conclusion (Appendix C) and recompute Retention on this subset. Table 3a compares it with the full benchmark, while Table 3b reports the output-change rate by error type.

The gold and polluted values are judged to lead to different task outputs for 146 of 167 slots (87.4%). Depending on the workflow, this means a different product recommendation, live-assistant intervention, topic suggestion, or other task result. The change rate is at least 82.6% for every non-numeric error type and 65.6% for numeric bucket shifts. In a typical non-changing numeric case, average watch times of 64 and 77 seconds both clear a 60-second threshold and therefore produce the same downstream result. Restricting evaluation to the 146 output-changing slots changes mean Retention only from 0.416 to 0.408. The near-identical Retention shows that F1's performance loss is not driven by factual differences that leave downstream behavior unchanged; it persists, and is slightly larger, when silent errors alter the e-commerce action or result.

Table 3: E-commerce task-output impact and result robustness.

Subset	Slots	Share	Ret.
All	167	100.0%	0.416
Output-changing	146	87.4%	0.408

(a) Overall output impact.

Error type	Slots	Change rate
Cat. granularity	17	100.0%
Cat. sibling	20	95.0%
Profile attribute	35	94.3%
Semantic synonym	40	92.5%
Entity near-duplicate	23	82.6%
Numeric bucket	32	65.6%

(b) Output-change rate by error type.

D2: Damage concentrates by error type and slot position. Damage varies across both silent-error types and slot positions: it is greatest for category errors (Table 4a) and for slots that occur early in the workflow trace (Table 4b).

Error type. Category errors are the most damaging error types in RITE: category granularity mismatch is hardest, category sibling drift is next, and numeric bucket shift is most noticeable but remains destructive (Table 4a. One possible explanation is that in the benchmark's recoverable environment, a wrong number or ID is checked against the producer response, whereas a wrong category appears anomalous only in context. For example, identifying "blockchain" as an implausible interest requires knowing the user watches local sports; the category value alone carries no anomaly.

Slot position. Early slots have the highest Noticed rate (0.264) but the lowest implicit-condition RSR (0.309) and Retention (0.399; Table 4b). One plausible explanation is that an early checkpoint yields a shorter prefix with fewer inherited observations, making its corruption easier to notice. At

the same time, the agent must complete a longer downstream path, giving the polluted value—or an incomplete recovery—more opportunities to affect later steps.

Table 4: Damage localization. Slots are paired units; Not. and RSR are measured in the implicit condition.

Error type	Slots	Not.	RSR	Ret.
Category granularity	17	0.099	0.140	0.180
Category sibling	20	0.121	0.199	0.292
Numeric bucket	32	0.342	0.224	0.297
Semantic synonym	40	0.231	0.307	0.382
Profile attribute	35	0.268	0.479	0.562
Entity near-duplicate	23	0.258	0.507	0.608

(a) Damage by silent-error type.

Slot position	Slots	Not.	RSR	Ret.
Early	128	0.264	0.309	0.399
Mid	26	0.175	0.384	0.452
Late	13	0.122	0.395	0.470

(b) Damage by slot position.

The positional trend therefore separates detection from final success: from early to late, Noticed falls, while implicit-condition RSR and Retention rise. This pattern makes upstream verification especially important.

D3: A lightweight intervention for handling silent errors. F2 shows that blind trust in inherited tool results is the dominant failure pattern, suggesting an additional reliability check during execution. We test State Checker, which asks the model to treat every tool result as potentially unreliable and, before using it, conduct an additional step to reflect on whether the result is trustworthy and should be verified. The underlying tools and benchmark instances remain unchanged. Claude-Opus-4.8, DeepSeek-V4-Pro, and GLM-5.2 each rerun half of their slots (Appendix C) with the additional check, paired against the same slots without it. State Checker increases Noticed and producer-refetch rates, but produces only modest gains in implicit-condition RSR (Figure 3).

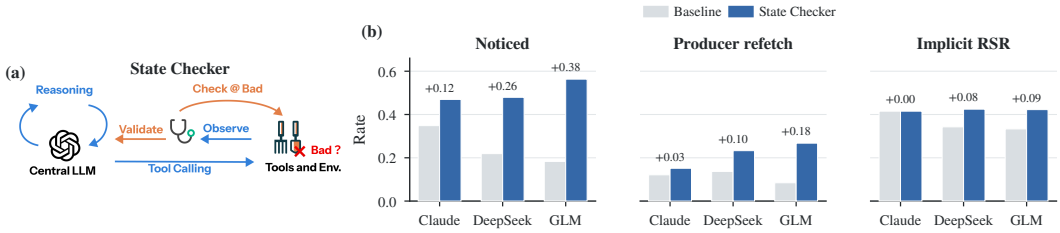


Figure 3: **(a)** State Checker adds a reliability check before a tool result is used. **(b)** Noticed, producer-refetch, and implicit-condition RSR with/without State Checker on paired implicit slots.

State Checker changes behavior in all three models, with larger increases in Noticed and producer refetch than in final RSR. The clearest behavioral shift is in GLM-5.2, whose producer-refetch rate more than triples; by contrast, Claude-Opus-4.8 gains no RSR despite noticing more errors.

State Checker makes evidence-seeking behavior steerable: an explicit reliability check increases recognition and use of the available verification path. The smaller RSR gains locate the remaining bottleneck in evidence integration and final task success, while the surfaced contradictions improve observability for a human or downstream controller. State Checker relies on the unpolluted verification path in RITE's single-site corruption setting. Persistent or system-wide tool faults may invalidate this path, requiring independent sources, tool substitution, redundancy, or human check.

7 Conclusion

We study silent errors in agentic tasks, where a plausible but incorrect tool result enters the execution history without an explicit failure signal and propagates through subsequent steps. We formulate a task that measures whether an agent can tolerate this corrupted state and still complete the original task, and introduce RITE, a benchmark built from 167 corruption slots across nine industrial e-commerce workflows grounded in real service logs. Across 10 recent LLMs, silent errors roughly halve task success and usually remain unnoticed; recovery remains incomplete even when models question the affected evidence and re-query its producer. Further analysis shows that these errors often alter downstream e-commerce outputs, with category errors and early-stage corruptions causing the greatest damage. Beyond the benchmark, we provide a simple reliability check that increases error detection and producer refetching more than final task success.

AUTHOR STATEMENTS

AI USE STATEMENT

In this work, we used generative AI tools in four roles. First, the authors wrote the initial draft of every manuscript section, after which AI tools were used to polish the language and improve clarity. Second, AI assisted benchmark construction by helping screen candidate workflows and production-log records (Appendix A) and by serving as model components during workflow execution. The authors manually reviewed every workflow selected through this process. Third, GLM-5 serves as the automatic judge of task success and observation state, and labels whether a corruption would change the business conclusion (Appendix C). Fourth, AI assisted with coding. The authors reviewed and validated all AI-assisted code, screening decisions, workflow outputs, and generated benchmark artifacts before use.

ETHICS STATEMENT

The workflows used in this study were deployed in production, and the logs were generated by their online operation. Before any log-derived data were provided to language models, sensitive identifiers such as user IDs were anonymized; benchmark runs used only anonymized data.

Benchmark data and generated outputs were isolated from production storage: no benchmark inputs, intermediate states, or outputs were reported to or written back to company production databases.

REPRODUCIBILITY STATEMENT

The paper documents the task, data sources, benchmark construction, evaluation protocol, dataset distribution, and auditable workflow examples needed to understand and assess the reported results.

The code is subject to an internal company review that may take several months. We will release it if approval is granted; otherwise, it will remain unavailable.

REFERENCES

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Chaoran Chen, Vy Nguyen, Ziji Zhang, Abhinav Gullapalli, Ziyi Wang, Yuxuan Lu, Dakuo Wang, Jing Huang, Zhou Yu, and Jin Lai. Retry, switch, or abstain? learning strategy-aware tool-use policies via controlled error injection, 2026. URL https://arxiv.org/abs/2608.11977.

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, January 2007. ISSN 1041-4347.

Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. Chat-rec: Towards interactive and explainable llms-augmented recommender system, 2023. URL https://arxiv.org/abs/2303.14524.

Bhaskar Gurram. Auditing automated evaluation, error propagation, and runtime mitigation in tool-using language agents, 2026. URL https://arxiv.org/abs/2604.16706.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. CRMArena-pro: Holistic assessment of LLM agents across diverse business scenarios and interactions. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=EPlpe3Fx1x.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, January 2025. ISSN 1558-2868. doi: 10.1145/3703155. URL http://dx.doi.org/10.1145/3703155.

Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. Recommender ai agent: Integrating large language models for interactive recommendations, 2024a. URL https://arxiv.org/abs/2308.16505.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=R0c2qtalgG.

Won Kim, {Byoung Ju} Choi, {Eui Kyeong} Hong, {Soo Kyung} Kim, and Doheon Lee. A taxonomy of dirty data. *Data Mining and Knowledge Discovery*, 7(1):81–99, January 2003. ISSN 1384-5810. doi: 10.1023/A:1021564703268.

Shirley Kokane, Ming Zhu, Tulika Manoj Awalgaonkar, Jianguo Zhang, Akshara Prabhakar, Thai Quoc Hoang, Zuxin Liu, Rithesh R N, Liangwei Yang, Weiran Yao, Juntao Tan, Zhiwei Liu, Shelby Heinecke, Huan Wang, Juan Carlos Niebles, Caiming Xiong, and Silvio Savarese. Toolscan: A benchmark for characterizing errors in tool-use LLMs. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=09tnQgqKuZ.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8EB8k6DdCU.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents, 2024. URL https://arxiv.org/abs/2401.13178.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1906–1919, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.173. URL https://aclanthology.org/2020.acl-main.173/.

Aritra Mazumder and Nusrat jahan Lia. Agentcheck: A reproduce-intervene-mitigate workbench for llm agents over mcp, 2026. URL https://arxiv.org/abs/2607.11098.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/25ae35b5b1738d80f1f03a8713e405ec-Paper-Conference.pdf.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Bo Peng, Xinyi Ling, Ziru Chen, Huan Sun, and Xia Ning. ecellm: Generalizing large language models for e-commerce from large-scale, high-quality instruction data, 2024. URL https://arxiv.org/abs/2402.08831.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation, 2025. URL https://arxiv.org/abs/2410.07869.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox, 2024. URL https://arxiv.org/abs/2309.15817.

Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. Tools fail: Detecting silent errors in faulty tools. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14272–14289, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.790. URL https://aclanthology.org/2024.emnlp-main.790/.

Shangyin Tan, Kevin Lin, Koushik Sen, and Matei Zaharia. Recovery-bench: Evaluating agentic recovery from mistakes. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025. URL https://openreview.net/forum?id=8FZRnDgDxq.

Liyan Tang, Tanya Goyal, Alex Fabbri, Philippe Laban, Jiacheng Xu, Semih Yavuz, Wojciech Kryscinski, Justin Rousseau, and Greg Durrett. Understanding factual errors in summarization: Errors, summarizers, datasets, error detectors. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11626–11644, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.650. URL https://aclanthology.org/2023.acl-long.650/.

Yang Tian, Zhengpeng Shi, Yu Zhou, and Bo Zhao. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability, 2026. URL https://arxiv.org/abs/2606.25819.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. Llms cannot find reasoning errors, but can correct them given the error location, 2024. URL https://arxiv.org/abs/2311.08516.

Wil M. P. van der Aalst. *Process mining: data science in action*. Springer, Germany, January 2016. ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. FreshLLMs: Refreshing large language models with search engine augmentation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13697–13720, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/

v1/2024.findings-acl.813. URL https://aclanthology.org/2024.findings-acl.813/.

Sri Vatsa Vuddanti and Satwik Kumar Chittiprolu. Recoverability has a law: The err measure for tool-augmented agents, 2026. URL https://arxiv.org/abs/2601.22352.

Haoxin Wang, Xianhan Peng, Huang Cheng, Yizhe Huang, Ming Gong, Chenghan Yang, Yang Liu, and Jiang Lin. ECom-bench: Can LLM agent resolve real-world E-commerce customer support issues? In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 276–284, Suzhou (China), November 2025. Association for Computational Linguistics. ISBN 979-8-89176-333-3. doi: 10.18653/v1/2025.emnlp-industry.19. URL https://aclanthology.org/2025.emnlp-industry.19/.

Jiangyuan Wang, Kejun Xiao, Qi Sun, Huaipeng Zhao, Tao Luo, Jian Dong Zhang, and Xiaoyi Zeng. Shoppingbench: A real-world intent-grounded shopping benchmark for llm-based agents, 2026a. URL https://arxiv.org/abs/2508.04266.

Ruipeng Wang, Yuxin Chen, Yukai Wang, Chang Wu, Junfeng Fang, Xiaodong Cai, Qi Gu, Hui Su, An Zhang, Xiang Wang, Xunliang Cai, and Tat-Seng Chua. Agentnoisebench: Benchmarking robustness of tool-using llm agents under noisy condition, 2026b. URL https://arxiv.org/abs/2602.11348.

Cailin Winston and René Just. A taxonomy of failures in tool-augmented llms. In *2025 IEEE/ACM International Conference on Automation of Software Test (AST)*, pp. 125–135, 2025. doi: 10.1109/AST66626.2025.00019.

Qian Xiong, Yuekai Huang, Ziyou Jiang, Zhiyuan Chang, Yujia Zheng, Tianhao Li, and Mingyang Li. Butterfly effects in toolchains: A comprehensive analysis of failed parameter filling in LLM tool-agent systems. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 16712–16729, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.907. URL https://aclanthology.org/2025.findings-emnlp.907/.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng YAN. Agentracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=l05DseqvuD.

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. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

Dongsheng Zhu, Xuchen Ma, Yucheng Shen, Xiang Li, Yukun Zhao, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in llm agents, 2026. URL https://arxiv.org/abs/2606.05806.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where llm agents fail and how they can learn from failures, 2025. URL https://arxiv.org/abs/2509.25370.

Yu Zhuang, Kefei Chen, Yitong Duan, Shuxin Zheng, Jian Li, and Xu-Yao Zhang. Agentrewind: Recoverable execution for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2608.14380.

A Workflow and Log Data Sources

This appendix documents two data-source stages: selecting the workflow collection $\mathcal{W}$ and mapping production logs to workflow inputs that yield eligible traces. It also records the resulting golden workflows. Workflow identifiers are anonymized and stable across the released artifacts.

Workflow selection: structural validity. We inspected e-commerce applications at the DSL level and retained complete deterministic DAGs with explicit control flow. We excluded graphs containing autonomous agentic nodes, atomic retrieval wrappers, short utility graphs, AB-test fragments, dangling references, or incomplete subgraphs. Fixed tool calls, read-only requests, code and LLM transformations, and explicit branch or loop nodes remained eligible.

Workflow selection: task diversity. Because the application library contains multiple versions of many task families, we deduplicated each family, selected a representative complete version, and formed a set spanning different business stages, task objectives, and data dependencies. These two workflow checks yielded nine golden workflows.

Log selection and trace retention. For each selected workflow, we mapped each candidate production-log record to the required start inputs and executed it on a structure-preserving replica. We selected a log only when its execution yielded a complete node-level trace with sufficient process depth and at least one business-semantic intermediate state that was produced upstream, consumed downstream, and locatable at a producer or consumer checkpoint. The corresponding trace was retained with the selected log. For safe trace collection, the replicas retained node identities, graph edges, data references, read-only retrieval, deterministic computation, and model-inference nodes, while writes, reporting calls, locks, memory updates, and external sends were neutralized or guarded.

Table 5 summarizes the selected workflows, their log-derived execution contexts, graph sizes, and resulting final units. Graph size reports the total nodes and statically counted tool- or model-facing interfaces in each collection replica. The interface count is descriptive and precedes trace-level eligibility checks. A *final unit* is one retained slot–locus pair before expansion into the three conditions.

Table 5: The nine golden workflows used by RITE. Log-derived inputs are described by semantic role rather than internal field names. Graph size is reported as total nodes / static tool-model interfaces.

ID	Task abstraction	Log-derived execution context	Graph size	Final units
wf01	Product recommendation from user needs, creator profiles, and selection history	Anonymized viewer, creator, and request context. A task query is constructed from the available profile context	55 / 14	11
wf02	Live-assistant scheduling and intervention selection	Anonymized session, creator, task, and configuration context	58 / 11	22
wf04	Pre-session script generation from creator and audience context	Anonymized creator identity. Profiles and historical content are retrieved by read-only nodes	22 / 11	15
wf05	Real-time product-presentation coaching	Anonymized session, creator, task, and available configuration context	47 / 12	17
wf06	Cross-domain intent and interest modeling	An anonymized user identity passing data-availability screening. Behavioral and profile records are retrieved downstream	52 / 14	52
wf07	Post-session summary over content, interaction, and commerce signals	Anonymized session, creator, request, and a session-derived time window	56 / 10	15
wf08	Topic suggestion from historical videos and comments	Anonymized creator, video or session, and request context	21 / 5	12
wf09	Extraction of reusable presentation patterns from strong sessions	Anonymized session, creator, request, and analysis window	37 / 5	9

Continued on next page.

Table 5 continued.

ID	Task abstraction	Log-derived execution context	Graph size	Final units
wf10	Periodic perception of the current live-session state	Anonymized session, creator, task, and available configuration context	46 / 9	14
Total: nine golden workflows			**394 / 91**	**167**

B Silent-Error Taxonomy and Literature Evidence

We synthesize 15 papers spanning agent and tool failures, data quality, and factuality. Two authors independently coded the error phenomena and boundaries described by each source, mapped overlapping concepts to a common vocabulary, and reconciled the mappings. The resulting taxonomy describes the semantic form of a plausible but incorrect intermediate value. Injection location is recorded separately.

B.1 Operational Definitions

The main taxonomy contains two semantic groups and seven leaves (Table 6). The first group changes which object or category is referenced; the second preserves the object while changing one of its attributes or values. Each leaf requires a schema-valid value whose task-relevant meaning differs from the recorded value.

Table 6: Operational definitions and boundaries of the seven silent-error leaves.

ID	Leaf	Operational definition	Primary boundary
L1.1	`category_sibling_drift`	Replace a category with a nearby category at the same hierarchy level.	Parent–child substitutions belong to L1.2.
L1.2	`category_granularity_mismatch`	Replace a category with an ancestor or descendant, making its scope too broad or too narrow.	Same-level substitutions belong to L1.1.
L1.3	`entity_near_duplicate`	Replace the intended entity with another real, similar entity from the candidate domain.	If entity identity is preserved and only an attribute changes, use L2.1.
L2.1	`profile_attribute_drift`	Preserve entity identity but replace a profile, preference, label, or other attribute with another valid value.	Whole-entity replacement belongs to L1.3.
L2.2	`numeric_bucket_shift`	Move a numeric value or interval to a neighboring valid range.	Malformed, impossible, or plainly out-of-range values are explicit failures.
L2.3	`semantic_synonym_drift`	Substitute text that appears semantically close but changes the business interpretation used downstream.	Meaning-preserving paraphrases are not errors.
L2.4	`temporal_staleness`	Reuse a value that was valid previously but is no longer correct at the task time.	A contemporaneous neighboring numeric value belongs to L2.2.

The injection locus is annotated independently of the semantic leaf and takes one of two values: `tool_output`, where the substitute is written into the producer observation, or `tool_input`, where it is written into a downstream tool argument.

B.2 Literature Grounding

Silentness and exclusion boundaries. *Tools Fail* (Sun et al., 2024) establishes the central operational criterion: a tool output may be schema-valid and wrong without raising an exception. The *Tool-Augmented LLM Failure Taxonomy* (Winston & Just, 2025) reinforces the distinction between explicit and silent failures. The *LLM Hallucination Survey* (Huang et al., 2025) and *Where LLM Agents Fail* (Zhu et al., 2025) further separate entity, context, profile, memory, and parameter errors, motivating value-level interventions while excluding failures that arise only from the agent's internal reasoning or memory. Other taxonomies delimit the scope from adjacent phenomena: the *Multi-Agent Failure Taxonomy* (Cemri et al., 2025) covers behavioral and system failures, τ-bench (Yao et al., 2025) emphasizes domain-policy compliance, CRMArena-Pro (Huang et al., 2026) adds confidentiality failures, and *Reasoning Error Localization* (Tyen et al., 2024) distinguishes recognizing an error from correcting its downstream effects. Together, these sources define silent error

as an incorrect intermediate value rather than a planning, collaboration, policy, confidentiality, or reasoning failure.

Semantic distinctions. Data-quality and factuality research supplies the distinctions used to organize the seven leaves. The *Dirty Data Taxonomy* (Kim et al., 2003) separates category, value, relation, and stale-data errors, supporting category-level substitutions, attribute and numeric changes, and temporal staleness. *Faithfulness versus Factuality* (Maynez et al., 2020) frames an error as a contradiction with its source evidence, providing a common criterion across all leaves. The taxonomy of *Factual Errors in Summarization* (Tang et al., 2023) distinguishes entity, attribute, numeric, and temporal errors, which map naturally to entity near-duplicates, profile-attribute drift, numeric-bucket shifts, and temporal staleness.

Specialized leaves and loci. Several sources refine particular boundaries. *Butterfly Effects in Toolchains* (Xiong et al., 2025) shows how incorrect tool arguments propagate downstream, motivating `tool_input` as an intervention locus in addition to `tool_output`. Work on *Duplicate Record Detection* (Elmagarmid et al., 2007) motivates the distinction between real near-duplicate entities and superficially similar text. *FreshLLMs* (Vu et al., 2024) grounds temporal staleness in answers that change over time. Finally, *LM-Emulated Sandbox Risks* (Ruan et al., 2024) helps exclude system and safety failures from the semantic taxonomy.

Taken together, the agent and tool literature defines when an error is silent and what lies outside scope, while the data-quality and factuality literature supplies the semantic forms of the corrupted value. We adapt these distinctions into schema-valid, single-slot workflow interventions; the semantic leaf describes what changed, and the locus records where that change enters the trajectory.

C Evaluation Protocol Details

This appendix specifies the rollout, judging prompts, and process measures used in evaluation. All prompts, trajectories, and case-study contents in this paper are English translations.

Evaluation flow. Each rollout follows four stages:

(i) **Initialize.** For benchmark unit i and condition $k \in \{\text{correct}, \text{implicit}, \text{explicit}\}$, the query is q_i, the continuation environment is E_i, and the condition-specific prefix p_i^k initializes the agent history as $h_{0,i}^k = p_i^k$. The correct prefix preserves the recorded value, the implicit prefix substitutes a silently incorrect value, and the explicit prefix presents an overt failure. The agent receives the task instruction and task inputs, followed by the prefix rendered as a numbered record of thoughts, actions, and observations.

(ii) **Execute.** The agent follows Eq. 1 for at most 25 steps. At each step, it outputs a thought and a JSON action that either calls a tool with arguments or stops with a final answer. Each condition uses a newly initialized copy of the same workflow-specific tool registry and case-specific replay snapshot. Tool names are normalized through workflow aliases. For each call, E_i first searches for an argument-matched recorded response and returns it if found; repeated unmatched calls may consume the next unconsumed response in recorded order, while calls absent from or exhausted in the snapshot fall back to the live read-only toolbox. Replay hits, misses, live fallbacks, and observation lengths are recorded for audit.

(iii) **Judge.** GLM-5 at temperature 0 independently applies the state-judge and result-judge prompts below. The same model also judges whether an agent that faithfully uses the gold versus the polluted value would reach a different business conclusion, which defines the output-changing slots in D1 (Section 6).

(iv) **Measure.** Producer refetch and Noticed are computed from the trajectory independently of both judge outputs.

Rollout counts and exclusions. Across the ten models, each condition has 1,670 scheduled rollouts; in total, 73 scheduled rollouts produced no valid record. A rollout is excluded if any of its observations contains a tool-execution error message, which indicates an environment-side anomaly; this rule removes 201 rollouts. The retained rollouts number 1,599, 1,575, and 1,562 for the correct,

implicit, and explicit conditions. Noticed is computed on all retained rollouts, and RSR on the 1,576, 1,555, and 1,543 retained rollouts that received a result judgment.

Judge prompts. Braced fields denote values filled separately for each benchmark instance.

State judge. The state judge determines whether the latest valid producer observation recovers the affected evidence.

> You are a strict evaluator. An intermediate slot may have been corrupted.
> **Slot:** {`slot`}; **condition:** {`condition`}.
> **Gold:** {`gold_value`}.
> **Polluted:** {`polluted_value`}.
> **Latest observation after the agent re-invoked the producer:**
> {`latest_observation`}.
> Classify the value as `gold`, `polluted`, `neither`, or `unknown`. Differences in formatting do not change semantic equivalence. Return only JSON with fields `used` and `reason`.

The outputs `gold`, `polluted`, and `neither` correspond to Recovered, Residual, and Neither observation states, respectively; `unknown` indicates insufficient evidence for attribution.

Result judge. The result judge determines whether the final answer completes the original user task.

> You are a strict evaluator. Decide only whether the agent's final answer solves the user task.
> **User task:** {`user_task`}.
> **Correct reference evidence:** {`correct_reference_evidence`}.
> **New observations:** {`new_observations`}.
> **Final answer:** {`final_answer`}.
> Check whether every task-critical claim is supported by either the reference evidence or a new observation. A contradiction unsupported by new evidence, an incomplete answer, or an error termination is unsolved. Return only JSON with fields `solved` and `reason`.

The `solved` output defines RSR. The state and result judgments are independent: recovering the affected evidence does not by itself guarantee a correct final answer.

Process measures. *Producer refetch* records whether the agent invokes the producer of the affected value again after that value appears in the prefix. *Noticed* is true when the reasoning explicitly signals a related content or result anomaly and is followed by a remedial action, such as refetching the producer, consulting a previously unused corroborating tool, or issuing an explicit refetch.

Table 7: Objects evaluated by the process measures and judge outputs. These quantities are complementary rather than successive scoring rules.

Measure	Evaluated object	Operational interpretation
Producer refetch	Post-prefix tool actions	Whether the agent re-invokes the producer of the affected value, regardless of what the new call returns.
Noticed	Reasoning plus remediation	Whether the agent explicitly identifies a related anomaly and follows it with a remedial action.
Observation state	Latest valid producer observation	Recovered matches gold, Residual matches the polluted value, and Neither matches neither reference value.
RSR	Complete final answer	Whether the answer resolves the original query using the available evidence; it is independent of observation-state attribution.

State Checker subset. The State Checker comparison in D3 (Section 6) uses 81 slots. We stratify slots by workflow and error type and take the first half of each stratum (rounded up), ordered by slot identifier. Workflow wf05 is excluded because one of its internal model components was being taken offline.

D HUMAN VALIDATION OF RESULT AND NOTICED LABELS

This appendix reports an independent human validation of two automatically produced evaluation labels: the LLM-based Result judgment that defines RSR and the trajectory-derived Noticed label.

Recruitment and assignment. We recruited six university participants through a campus forum. The recruitment post described an approximately 50-minute annotation task, and each participant received RMB 50 upon completion. We selected 100 rollouts using stratified sampling: 28 were from the correct condition, 35 from the explicit-error condition, and 37 from the implicit-error condition, with 10 rollouts from each evaluated model. Within each stratum defined by model and condition, rollouts were drawn uniformly at random. Each rollout was independently assigned to three participants, and each participant annotated 50 rollouts, yielding 300 participant–rollout judgments. To balance overlap, each of the 20 possible three-person annotator groups was assigned five rollouts. Participants were instructed to work independently and not discuss individual cases.

Two-stage annotation protocol. For each rollout, the interface presented the user task, the correct reference evidence, the complete agent trajectory, any new tool observations, and the final answer. Annotation then proceeded in two stages. In the blind stage, participants answered two binary questions: (i) whether the final answer solved the user task (Result), and (ii) whether the agent explicitly noticed a data anomaly and subsequently invoked a tool to verify it (Noticed). A Noticed label therefore required both an explicit recognition signal in the reasoning and a later verification action. Once submitted, these labels were locked. In the review stage, the interface revealed the two automatic evaluation labels and their rationales. Participants marked each label as *agree*, *disagree*, or *unsure*, with an optional comment. Figure 4 illustrates the two stages. Because the second-stage responses may be influenced by the revealed rationales, we treat the blind-stage comparison as the primary validation and report review-stage agreement only as complementary evidence.

Agreement results. For the blind-stage analysis, we used the majority vote of the three independent annotations as the human reference label for each rollout. Table 8 reports raw agreement and Cohen's κ between the human reference and the corresponding automatic label, as well as Fleiss' κ among the three human annotators. Cohen's κ adjusts raw agreement for the agreement expected from the two label distributions. Precision, recall, and F1 treat the human reference as the target and the positive automatic label as the prediction. Review agreement is the fraction of all 300 participant–rollout responses marked *agree* after the automatic labels were revealed.

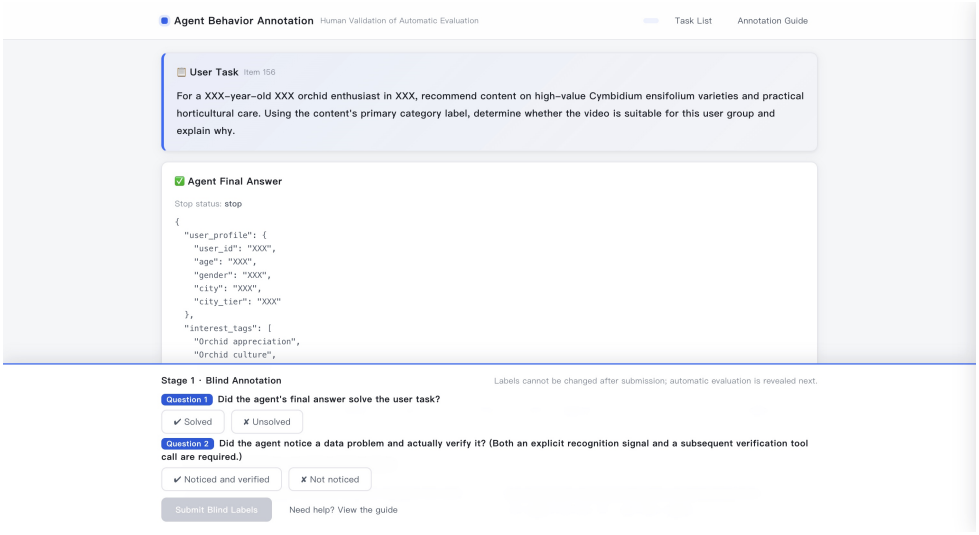
Table 8: Human validation of the automatic Result and Noticed labels over 100 rollouts. Each rollout received three independent annotations. Blind-stage statistics provide the primary validation; post-reveal agreement is complementary.

Label	Majority agreement	Cohen's κ	Human Fleiss' κ	Precision	Recall	F1	Review agreement
Result	76/100 (76%)	0.318	0.477	97.1%	75.3%	84.8%	252/300 (84%)
Noticed	87/100 (87%)	0.739	0.569	87.0%	85.1%	86.0%	267/300 (89%)

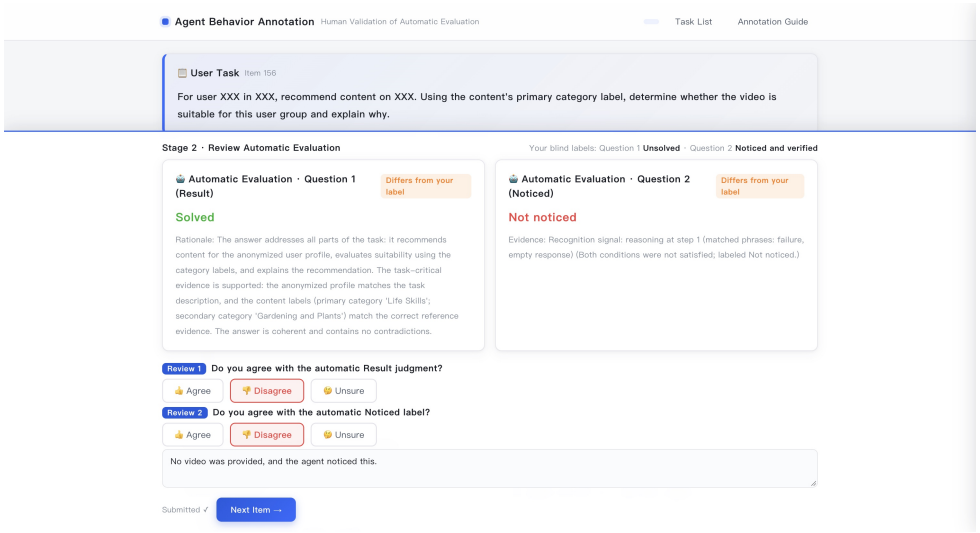
The trajectory-derived Noticed label aligns strongly with the human reference (87/100; Cohen's $\kappa = 0.739$). Result agreement is lower (76/100; $\kappa = 0.318$), but its high precision (97.1%) and lower recall (75.3%) indicate a conservative evaluator: positive success judgments are usually supported by the human majority, while some human-accepted rollouts are labeled unsolved.

The asymmetric Result errors help explain its lower κ: the human majority labeled 89/100 rollouts as solved, compared with 69/100 for the result judge, and 22 of the 24 disagreements followed this direction. Inspection of the rationales and comments suggests that participants sometimes accepted broadly responsive answers, whereas the result judge more strictly enforced query-specific completeness and evidence support. This strictness follows the judge's design: an answer counts as solved only if every task-critical claim is supported by the reference evidence or a new observation, and the same judge and criterion are applied to all three matched conditions. Although human Fleiss' κ for Result is 0.477, all three participants agreed on 80/100 rollouts, indicating that disagreement

is concentrated in a limited set of borderline cases rather than reflecting widespread annotation instability. After the automatic labels and rationales were revealed, participants agreed with 84% of the Result judgments and 89% of the Noticed labels. Because these responses may be influenced by the revealed rationales, we do not treat them as independent evidence of accuracy; instead, they show that most automatic decisions were acceptable to participants once explained. Taken together, the blind and review stages provide more consistent support for Noticed, while Result disagreement mainly reflects the evaluator's stricter success criterion.



(a) Blind annotation. Automatic evaluation labels are unavailable, and the participant independently labels Result and Noticed.



(b) Post-reveal review. The interface shows each automatic evaluation label and its rationale for agreement assessment.

Figure 4: English translation of the original Chinese two-stage annotation interface used in the human-validation study. Blind labels were locked before the automatic evaluation labels and rationales were revealed.

E End-to-End Case Studies of Silent-Error Handling

This appendix presents two baseline case studies, not State Checker interventions. In Case 1, the agent successfully refetches the corrupted producer but later overwrites the recovered evidence dur-

ing reasoning. Case 2 follows an Unnoticed implicit error from a plausible category substitution to a wrong final recommendation.

How to read the case boxes. Entries labeled **P*** summarize inherited prefix events shown to the agent; entries labeled **R*** are continuation steps generated by the evaluated model. Each step lists the agent action, the observed result, and a diagnostic note. *Producer refetch* records whether the model re-invokes the producer of the affected value. *Observation state* describes the latest valid producer observation as Recovered, Residual, or Neither, while *final task success* separately asks whether the complete answer resolves the user query.

E.1 Case 1: Producer Refetch Succeeds but Final Recovery Fails

The setup box makes the instance, intervention, and expected evidence explicit before presenting the trajectory.

Case 1: Setup

Model and condition: Qwen3.6-35B; baseline implicit condition

Workflow: wf09, live-stream assistant-data extraction

Affected producer: BuildCommentRequest

User task: Extract parent interactions and assistant speech from the opening warm-up stage, then use that stage's end time to determine the sentiment of the final interaction.

Task input: `feed_id`=*anonymized*; `start_timestamp`=1787916387; `end_timestamp`=1787920006

Gold slot value: `start_sec`=1787916387; `end_sec`=1787920006

Implicit-prefix value: `start_sec`=1787916527; `end_sec`=1787920146

Why the error is silent: The shifted interval has the expected schema and plausible timestamps, and the prefix contains no warning or competing value.

Case 1: Prefix and Rollout

Inherited prefix supplied to the model

P1 Call GetFeedDashboard.
Result: The recorded observation is empty.
Diagnosis: No useful stage evidence is established before the affected slot.

P2 Call BuildCommentRequest with the original feed identifier and timestamps.
Result: The prefix reports `start_sec`=1787916527 and `end_sec`=1787920146.
Diagnosis: Silent corruption enters the history as a plausible request interval.

Model-generated continuation

R1 Reuse the inherited request in FetchAllCommentsRPC.
Result: The RPC returns an argument error because the inherited request omits the feed identifier.
Diagnosis: A downstream explicit error appears, but the shifted interval itself is not yet corrected.

R2 Attribute the failure to the missing identifier and refetch BuildCommentRequest with the task-level timestamps.
Result: The tool returns the feed identifier with `start_sec`=1787916387 and `end_sec`=1787920006.
Diagnosis: Producer refetch succeeds and restores the gold slot value.

R3 Fetch comments with the corrected request.
Result: The call completes. Boundary records include 1787919998 ("Can Teacher J. dance?"), 1787920002 ("Teacher J."), and 1787920006 ("OK, OK, OK").
Diagnosis: Correct downstream evidence is acquired through the recovered end time.

R4 Construct an ASR request using the corrected interval.
Result: The request preserves the recovered start and end boundaries.
Diagnosis: The model initially carries the repaired evidence into a second task branch.

Case 1: Prefix and Rollout (continued)

R5 Invoke the ASR RPC.
Result: The call completes but returns no speech payload.
Diagnosis: Assistant-speech evidence remains unavailable, increasing pressure to use a fallback.

R6 Parse the full comment sequence.
Result: The parser reports 482 records, including early comments and final records at 1787920002 and 1787920006.
Diagnosis: The recovered boundary remains supported by observable comments.

R7 Invoke candidate segmentation for the opening stage.
Result: The tool returns no candidate segments.
Diagnosis: Missing stage annotations trigger fallback reasoning, not evidence that the recovered interval is wrong.

R8 Reparse a compact boundary set.
Result: Ten boundary comments are returned, again including the final interaction at 1787920006.
Diagnosis: Correct endpoint evidence remains available immediately before the decisive error.

R9 Replace the recovered boundary with a self-defined ten-minute warm-up window.
Result: The model selects the earlier "[strong]" reaction at 1787916453 and labels it positive.
Diagnosis: **Reasoning overwrite:** a heuristic supersedes recovered evidence and causes the wrong final interaction.

Reported answer

The model reports "yeah" and "[strong][strong][strong]" as the opening-stage parent interactions, states that assistant speech is unavailable, and identifies "[strong][strong][strong]" as the final interaction with positive sentiment. The answer is not copied directly from the polluted prefix; it fails after the interval has already been repaired.

Case 1: Judgment

Producer refetch: Yes. R2 re-invokes BUILDCOMMENTREQUEST.

State judge: Recovered. R2 returns both task-provided timestamps exactly.

Final task success: Unsolved. Later comments exist through 1787920006, but the answer selects an earlier interaction from a heuristic window.

Observation recovery and final task success intentionally evaluate different objects.

State progression: Prefix (plausible corrupted interval) $\rightarrow$ verification (correct interval) $\rightarrow$ downstream use (correctly fetched comments) $\rightarrow$ reasoning (heuristic ten-minute window) $\rightarrow$ answer (wrong final interaction). The failure occurs after successful evidence reacquisition.

Failure mechanism. The decisive error is a *reasoning overwrite*. The model obtains and initially uses the correct interval, but later treats missing stage annotations as permission to invent a generic boundary. Producer refetch is therefore necessary but insufficient: recovery also requires preserving the authority of corrected evidence over later derived decisions.

E.2 CASE 2: AN UNNOTICED CATEGORY ERROR PROPAGATES TO THE ANSWER

The second case shows the complementary failure mode: the model never questions or refetches the affected producer, so the polluted value remains the organizing premise throughout the continuation.

Case 2: Setup

Model and condition: GLM-5.2; baseline implicit condition
Workflow: wf06, cross-domain intent and interest modeling
Affected evidence: A content category in the user's article-reading history

Case 2: Setup (continued)

User task: Use interests in CATL news and AI side-income content to analyze the profile, generate live-stream recommendation keywords, and choose directly between the finance and pet domains.

Gold category: First level: Finance; second level: Stocks

Implicit-prefix category: First level: Pet; second level: Cat

Unchanged context: Article identifier, timestamp, scene, and `read_time`=115920 remain unchanged; the model sees no competing Finance/Stocks value.

Why the error is silent: Pet/Cat is well formed and locally plausible, and no warning accompanies the article record.

Case 2: Prefix and Rollout

Inherited prefix supplied to the model

P-context Receive the normal profile and live- and short-video behavior history.
Result: The user is a middle-aged man in a large city; no prior record establishes a pet-domain preference.
Diagnosis: Background context does not independently support the later Pet/Cat conclusion.

P-error Receive an article record with `category1=Pet`, `category2=Cat`, and `read_time`=115920.
Result: The record is schema-valid and carries no error signal.
Diagnosis: A Finance/Stocks record is silently replaced by Pet/Cat at the affected slot.

Model-generated continuation

R1 Treat the inherited category and long reading time as strong evidence of pet interest; request short-video summaries.
Result: The only positive pet evidence is the polluted article category.
Diagnosis: The corrupted value becomes the initial domain hypothesis.

R2 Interpret comedy, daily-life, and public-speaking summaries as showing no finance or AI interest; classify the user as pet-domain and request recommendation terms.
Result: The new summaries contain no direct pet evidence.
Diagnosis: **Confirmation through absence:** non-finance evidence is misread as positive support for Pet/Cat.

R3 Accept the encoder output and stop.
Result: The encoder returns the user's city, pet snacks, canned cat food, pet food, cat litter boxes, cat litter, and related terms.
Diagnosis: A category-level error is concretized into product-level recommendations.

Reported answer

The final answer classifies the user as belonging to the pet domain, cites the Pet/Cat article record with `read_time`=115920 as its primary justification, and recommends pet-food and cat-supply keywords.

Case 2: Judgment

Noticed: No. No thought questions Pet/Cat or requests verification of the article record.

Producer refetch: No. No continuation step revisits the source of the article category.

Observation state: Residual. The model continues to rely on the inherited Pet/Cat record.

Final task success: Unsolved. The pet-domain conclusion contradicts Finance/Stocks, and no new observation supports Pet/Cat.

State progression: Prefix (Finance replaced by Pet) $\rightarrow$ adoption (Pet treated as strongest interest) $\rightarrow$ interpretation (absence of finance videos used as confirmation) $\rightarrow$ tool output (pet-product keyword generation) $\rightarrow$ answer (wrong domain recommendation). The polluted value shapes every later interpretation.

Failure mechanism. The important transition occurs in R2. The additional evidence is non-financial, not positively pet-related, but the inherited category has already established a pet hypothesis. The model therefore interprets absence of finance content as confirmation rather than insufficient evidence. This is the propagation pattern targeted by the benchmark: a locally plausible error becomes an organizing premise, shapes later evidence interpretation, and determines the recommended action.

F Dataset Distribution

This appendix reports the distribution of the 167 paired units underlying Section 4.3. Each unit is one retained slot–locus pair from a clean workflow execution and expands into three conditions, yielding 501 instances. Because one execution may contribute multiple slots or loci, all counts below refer to paired units rather than workflow runs.

Figure 5 shows the distribution by workflow on the left and by silent-error type on the right. Both panels use the same vertical scale and label every bar with its exact count.

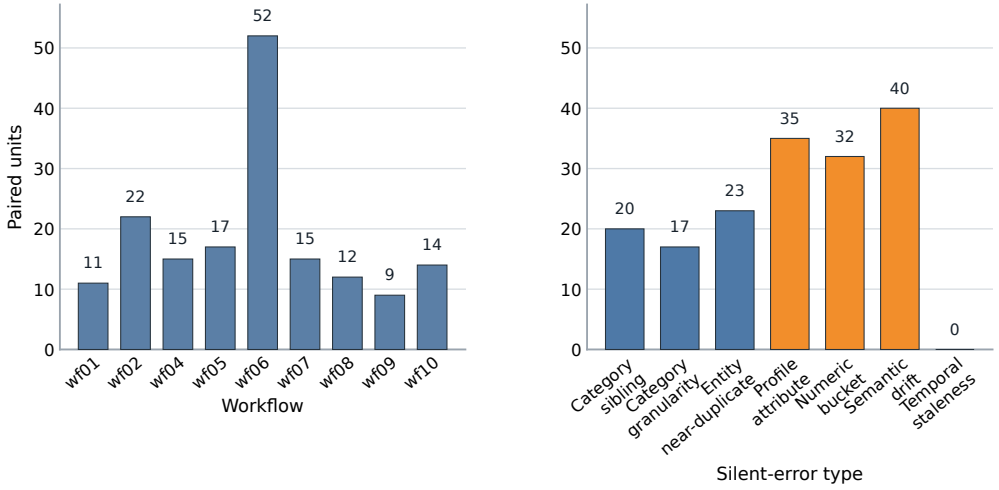


Figure 5: Distribution of the 167 paired units. **Left**: units contributed by each workflow. **Right**: units per silent-error type; blue bars denote object or category errors and orange bars denote attribute or value errors. Numeric labels give exact counts, and temporal staleness has no retained instance.

All nine workflows are represented: wf06 contributes 52 units, and each remaining workflow contributes between 9 and 22. Of the 167 units, 147 place the intervention at `tool_output` and 20 at `tool_input`.

Across error types, semantic synonym drift, profile attribute drift, numeric bucket shift, entity near-duplicate, category sibling drift, and category granularity mismatch contribute 40, 35, 32, 23, 20, and 17 units, respectively. Temporal staleness has no retained instance.

These counts account for all 167 paired units and provide the denominators for per-workflow and per-type analyses.

G Limitations

Several limitations qualify our findings. First, each silent-error instance contains a single corrupted value, and re-querying its producer returns the unpolluted response; RITE therefore measures recovery when a trustworthy verification path is available, and faults that persist across re-queries are left to future work. Second, recorded calls are replayed, whereas valid calls absent from the recordings are served by read-only live tools, so continuations that leave the recorded paths may observe current tool behavior. Third, temporal staleness is defined in the taxonomy but not instantiated. Finally, RSR relies on an LLM judge, whose errors may affect absolute scores.