When Tool Outputs Go Wrong: Belief Revision in Tool-Using Agents

Avni Mittal [1]

avni.mittal2002@gmail.com

Abstract

Tool-using agents depend on intermediate observations that may be stale, irrelevant, or simply wrong, yet they are usually evaluated only by their final answer. This hides a central failure mode of long-horizon behavior: corrupted evidence can enter early in a trajectory and propagate downstream without being challenged. We study this problem as belief revision under contradicting evidence. Starting from successful ReAct trajectories on HotpotQA, we replace one essential tool output with a controlled perturbation, then resume the run with honest tool calls. Across 3,175 perturbed runs on three models and six perturbation types, recovery is common but mostly silent: 87.5% of judged runs still reach the correct final answer, while 54.8% do so without explicitly acknowledging that earlier evidence was wrong. Model choice is the main driver of robustness. `gpt-4.1` recovers in 94–98% of cells, while `Llama-3.3-70B-Instruct` recovers in 71–78%. Missing or obviously irrelevant evidence is often noticed, but fluent wrong evidence usually is not.

1. Introduction

Tool-using agents operate in trajectories where each step depends on observations produced by tools, retrieval systems, or memory written earlier in the run (Yao et al., 2023; Schick et al., 2023). Those observations are not guaranteed to be correct. Search may return the wrong paragraph, a retrieved document may be irrelevant, an API may surface stale information, and a tool failure may silently produce a fluent but wrong result that the agent trusts (Sun et al., 2024). Once such evidence enters the context, the rest of the trajectory can inherit it, and early errors are known to cascade and compound over later steps (Zhu et al., 2025).

The dominant evaluation paradigm scores only the endpoint of a run, whether an exact-match final answer or a terminal environment state (Mialon et al., 2024; Yao et al., 2024). That is useful for measuring end-to-end task success, but it cannot distinguish a trajectory that goes wrong because the model reasons poorly from one that goes wrong because an earlier tool output was wrong and the agent never revised its belief after later evidence contradicted it. These are different failure modes, yet endpoint scoring conflates them, and it also hides the substantial mid-trajectory drift that noisy tool outputs induce (Wang et al., 2026).

This paper studies that second case, which we call belief revision under contradicting evidence. When a tool-using agent receives a wrong output early in a trajectory and later honest tool outputs disagree with it, what does it do: detect the contradiction, recover to the right answer, perform extra verification, or quietly continue with the corrupted belief? Final accuracy cannot say. A correct final answer may reflect careful cross-checking or a silent shift that leaves no trace in the reasoning (Turpin et al., 2023); a wrong answer may reflect pure propagation or a contradiction the agent noticed but failed to resolve.

We study this with a controlled counterfactual: take a trajectory that succeeded honestly, replace one essential tool output, and observe the resumed run. Because the clean and perturbed runs share the same task, prefix, and later opportunity for correction, this isolates belief revision more sharply than aggregate ablations. We instantiate it on HotpotQA's distractor setting (Yang et al., 2018), whose supporting-fact annotations make intervention-point selection precise, applying one of six controlled perturbations (from empty returns to fluent, authoritative-sounding wrong paragraphs) at a single tool call and leaving every later call honest. The main contributions are:

- We introduce a controlled counterfactual protocol for studying how tool-using agents respond to corrupted intermediate evidence while holding task, prompt, model, and trajectory prefix fixed.
- We provide an operational outcome taxonomy that separates recovery from explicit noticing, making trace-

[1]Independent Researcher. Correspondence to: Avni Mittal <avni.mittal2002@gmail.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

level behavior measurable rather than inferred from final-answer accuracy alone.

- We show that robustness must be evaluated along two distinct axes, recovery and detection, because agents can reach correct answers without explicit revision behavior.
- We analyze when revision-style behavior is more or less likely and evaluate a simple prompt-level mitigation that encourages explicit cross-checking before final answers.

2. Related Work

This paper connects four lines of work: agent failure analysis, knowledge conflict in retrieval-augmented generation, adversarial or unreliable tool outputs, and counterfactual analysis of LLM reasoning. The common thread is that end-to-end accuracy is not enough when the object of study is a multi-step trajectory.

2.1. Agent Failure Analysis and Recovery

The closest methodological neighbors are recent studies of agent failure attribution and recovery. Prior work uses trace-level interventions to identify which action caused a failure or how an agent can be trained to recover from broken tool calls (Zhu et al., 2025; Zhang et al., 2025b;a; Vuddanti et al., 2025). Our setup differs in both direction and locus of intervention. Instead of correcting a wrong action inside a failed run, we corrupt a correct tool output inside a successful run and then observe whether later honest evidence triggers belief revision. This makes the dependent variable different as well: we are not measuring attribution accuracy or post-hoc repair, but explicit detection, recovery, and verification behavior under contradicting evidence. This connects to work on self-correction, where models often fail to revise their own answers without an external signal and can even degrade when prompted to try (Huang et al., 2024), and to evidence that whether a model updates to new information or holds to a prior belief is itself task-dependent (Sun et al., 2025).

2.2. Knowledge Conflict in RAG

The closest substantive literature is work on knowledge conflict and noisy retrieval in RAG. Studies such as WikiContradict, MAGIC, and RAGuard show that LLMs struggle when retrieved passages disagree or when retrieval is subtly misleading (Hou et al., 2024; Lee et al., 2025; Zeng et al., 2025). We borrow that insight and some perturbation intuitions, especially entity and attribute substitutions. The difference is that those settings are largely single-shot: the model reads a static context and answers once. Our setting is agentic and multi-step. The wrong evidence appears at one specific tool call, and contradicting evidence can emerge later through the agent's own follow-up searches. That lets us study belief revision across time rather than conflict resolution inside one frozen context window.

2.3. Indirect Prompt Injection and Tool Reliability

Our perturbation surface also overlaps with work on indirect prompt injection and adversarial tool outputs (Greshake et al., 2023; Zhan et al., 2024; Debenedetti et al., 2024; Zou et al., 2024). That literature asks whether attacker-controlled retrieved content can steer an agent away from user intent. We ask a different reliability question: what happens when an otherwise honest tool returns a fluent but wrong intermediate result? This distinction matters because our perturbations are designed to resemble ordinary tool failures such as wrong entities, wrong attributes, empty returns, or plausible but irrelevant paragraphs, not attack strings or instruction hijacks.

2.4. Counterfactual Reasoning Analysis

Finally, our experimental logic follows the counterfactual tradition used to study chain-of-thought faithfulness and related reasoning phenomena (Lanham et al., 2023; Paul et al., 2024). Those works intervene on the model's own intermediate reasoning and test whether the final prediction changes, whereas we instead intervene on the output evidence the tools return. In that sense the paper studies environmental faithfulness rather than reasoning faithfulness: when the world presented through tools becomes inconsistent, does the agent detect the inconsistency and revise its belief? This perspective is also related to sycophancy work, where models sometimes abandon a correct answer under external pressure (Sharma et al., 2023; Wei et al., 2023). Our *NOTICED_BUT_FAILED* cases have a similar structure, except the competing signal comes from a prior tool observation rather than a user's stated belief.

3. Methodology

Figure 1 summarizes the experimental logic before we describe each component in detail. The design keeps the task, model, and trajectory prefix fixed, changes exactly one essential tool observation, and then resumes the run with honest later tool calls so that any difference in behavior can be attributed to belief revision under contradicting evidence.

3.1. Task and Agent

We use a ReAct-style agent (Yao et al., 2023) with one tool, `search(query)`, over the ten-paragraph distractor pool attached to each HotpotQA example. Retrieval is TF-IDF over paragraph title plus concatenated sentences, ranked by cosine similarity. Each tool call returns the top two para-

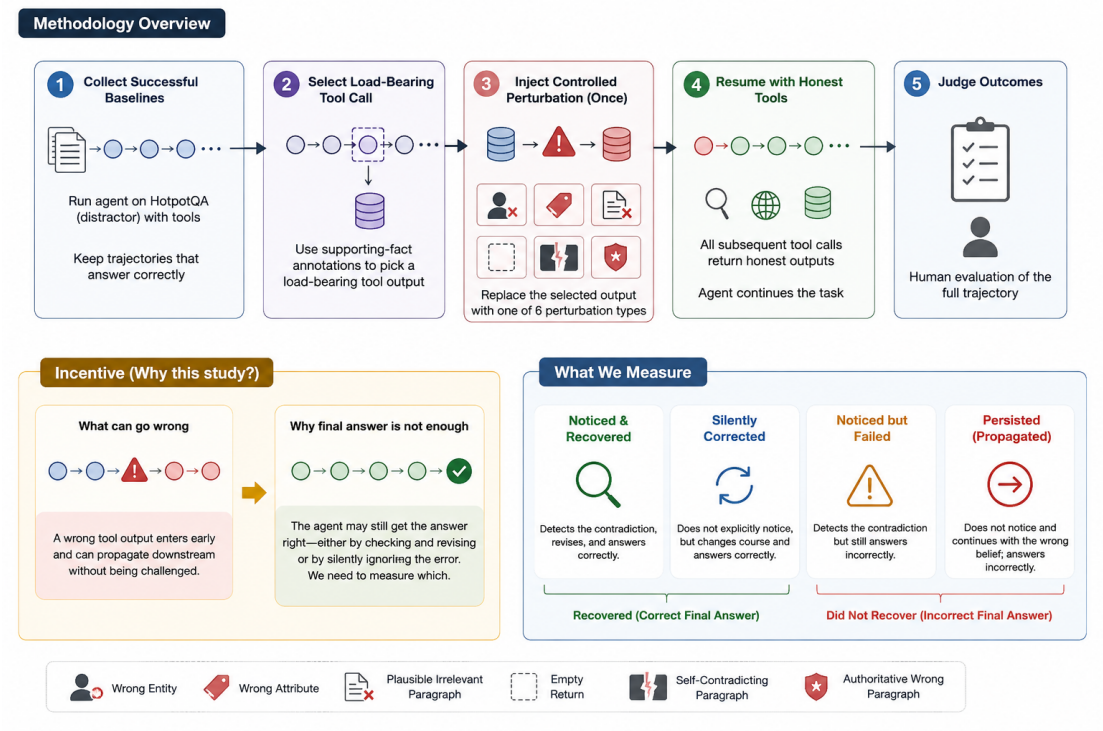


Figure 1. Methodology overview. We begin from successful clean trajectories, use supporting-fact annotations to select an important tool call, replace that single observation with one of six controlled perturbations, resume the trajectory with honest later tool outputs, and then code the resulting behavior into four outcome classes. The key incentive is to separate belief revision from other changes: the task, prompt, model, and trajectory prefix are held fixed, while only one intermediate observation is altered.

graphs formatted as a title followed by numbered sentences. The agent receives a maximum of eight steps and must emit either a tool call or a short final answer at each step. The verbatim system prompt is reproduced in Appendix B.1, and the runtime wrapper that appends each tool observation back into the context is in Appendix B.2.

The agent setup is intentionally simple. The goal is not to maximize benchmark performance through tool engineering. The goal is to isolate a single question about trajectory dynamics: if one tool output is corrupted early in the run, does the agent later revise its belief when honest evidence disagrees with it?

3.2. From Clean Trajectories to Intervention Points

The study starts with successful baseline trajectories. We keep a baseline run only if it has exact-match score 1, uses at least two search calls, terminates with a final answer, and contains at least one tool output with an annotated supporting fact. These conditions remove trivial one-hop or already-failed trajectories.

We identify candidate intervention points using three filters. First, the selected tool output must contain at least one supporting-fact sentence. Second, the content of that supporting fact must surface later in the trajectory, either in the agent's reasoning, a later query, or the final answer. Third, the selected tool call must occur early enough that at least two more steps remain in the trajectory. Together these filters aim to isolate tool outputs that the trajectory actually depends on and that can later be contradicted by honest evidence.

3.3. Perturbation Taxonomy

Each intervention point can be paired with up to six perturbation types. Table 1 summarizes both the perturbation taxonomy and the outcome classes used later in the analysis.

The taxonomy varies the kind of wrongness while holding the surface interface fixed. The agent still sees something that looks like a normal tool observation. What changes is whether the corruption is absent, off-topic, internally inconsistent, or fluent and plausible.

3.4. Counterfactual Replay

For each intervention point and perturbation type, we replay the original trajectory prefix up to the chosen tool call, replace the tool output with the perturbed observation, and resume the agent with honest tool calls for the remainder of the run. The prompt and the retriever remain unchanged.

Table 1. Methodology taxonomy: perturbation types used during replay and outcome classes used during evaluation.

Label	Description
Perturbation types	
WRONG_ENTITY	Replace the answer-relevant entity with a plausible same-type substitute.
WRONG_ATTRIBUTE	Keep the entity fixed and perturb a date, count, role, or relation.
PLAUSIBLE_IRRELEVANT	Swap in a topically adjacent paragraph that does not contain the supporting fact.
EMPTY	Return no result for the selected tool call.
SELF_CONTRADICTING	Insert a sentence that explicitly contradicts the original supporting fact.
AUTHORITATIVE_WRONG	Apply an entity substitution and add confident framing to the incorrect statement.
Outcome classes	
SILENTLY_CORRECTED	Final answer is correct, but the agent never explicitly notes the earlier corruption.
NOTICED_AND_RECOVERED	Final answer is correct and the trajectory contains an explicit detection signal.
NOTICED_BUT_FAILED	The agent flags a problem but still ends with the wrong answer.
PERSISTED	Final answer is wrong and the trajectory never acknowledges the corruption.

The agent is not told that the run is a replay.

This procedure is the key methodological choice. It preserves as much of the original run as possible while intervening at a single point. The replay therefore answers a counterfactual question about that trajectory, not about a different prompt or a different environment.

3.5. Outcome Coding

Each perturbed trajectory is coded into one of four outcome classes, listed in Table 1.

Detection is the axis that captures whether the agent *checked the corrupted evidence again* rather than silently moving on, and it is judged from the post-intervention reasoning trace alone. We operationalize it with a single LLM-judge task: the judge reads the agent's later thoughts, queries, and final answer, and returns a strict-JSON `detected` flag together with a short supporting quote. A trajectory is marked detected only when the trace exhibits one of three re-checking signals: (i) an explicit acknowledgement of inconsistency, contradiction, or doubt about earlier evidence (for example, "the previous result said X but this says Y" or "let me re-check"); (ii) re-querying a fact that a previous tool call already returned, with language indicating verification rather than routine progress; or (iii) re-reading or re-evaluating an earlier tool output. Mere additional searches without expressed doubt, routine continuation of the trajectory, and simply emitting a final answer that differs from the perturbation are explicitly *not* counted as detection. This binary detection label is the only judgment the LLM judge produces; answer correctness is scored separately by exact match, so the judge never sees or affects the recovery metric. The detection flag is what distinguishes the *NOTICED* classes from *SILENTLY_CORRECTED* and *PERSISTED* in Table 1. The exact judge prompt, including the full list of positive and negative detection signals, is reproduced verbatim in Appendix B.4. The main coding pass uses `gpt-4.1` as this strict JSON judge over all 3,175 perturbed runs; a later robustness check re-runs the identical prompt with `DeepSeek-V3.1` as an alternate judge on a stratified sample of 300 trajectories.

Our qualitative audit of sampled trajectories (Appendix A) helped sharpen how these labels should be interpreted in traces, not just in final answers. In *NOTICED_AND_RECOVERED* cases, the agent usually verbalizes a mismatch (for example, that a name, year, or title "does not seem accurate") and then issues a targeted follow-up query to repair the corrupted fact. In *SILENTLY_CORRECTED* cases, the final answer is also correct, but the trace typically moves onto an unaffected evidence path without ever explicitly flagging the earlier corruption. This distinction is why Table 1 separates recovery from explicit noticing.

The failure classes are behaviorally distinct as well. In *NOTICED_BUT_FAILED* trajectories, the agent often recognizes that something is off, but then overgeneralizes, keeps searching without resolving the key attribute, or outputs an answer that is too broad. In *PERSISTED* trajectories, the agent usually treats the perturbed observation as usable evidence, or fills the gap from prior expectations after an empty or misleading retrieval. These qualitative patterns make the coding scheme operational: the labels mark different revision behaviors, not only different end states. Appendix A gives representative trajectory excerpts in a format that makes the four behaviors easy to compare.

4. Experimental Setup

4.1. Dataset Construction

We use the HotpotQA distractor development split. The initial pool contains 7,405 questions. We keep only bridge questions, remove examples with more than four supporting facts, remove examples with fewer than nine context paragraphs, and remove questions whose answer is a bare number. This yields 5,370 eligible questions, from which we draw a seeded random sample of 400. The sample is fixed and reused across all downstream analyses. Table 6 in Appendix E.1 summarizes the full study funnel, from the

raw split through to the judged perturbed runs.

Two features of this dataset matter for the study. First, each example comes with a fixed paragraph pool, which makes replay deterministic. Second, supporting-fact annotations let us identify exactly which retrieved sentences were important in the clean run.

4.2. Models and Infrastructure

We evaluate three models through a single common harness: `gpt-4.1`, `DeepSeek-V3.1`, and `Llama-3.3-70B-Instruct`. All runs use temperature 0 and at most eight ReAct steps. The main replay study uses the same prompt, tool surface, and retrieval mechanism for all models.

Before the full collection run, small-scale calibration runs confirmed that the harness parsed all three model families correctly, surfaced a content-filter edge case, and produced stable per-model exact match. These early runs already indicated that `Llama-3.3-70B-Instruct` would yield fewer usable baseline trajectories because it often terminated after a single search.

4.3. Metrics

The study tracks exact match, F1, recovery rate, catastrophic failure rate, contradicting-evidence rate, number of post-intervention tool calls, and explicit detection rate. Recovery concerns the final answer only and is computed by exact match. Detection concerns whether the reasoning trace acknowledges a conflict or re-checks an earlier tool output, and is the LLM-judge label defined in §3.5 using the prompt in Appendix B.4. We report both because the paper's main claim depends on their separation.

5. Results

5.1. Baseline Performance and the Pool of Usable Trajectories

For the full baseline collection, each of the 400 candidate questions is run on all three models, giving 1,200 trajectories. Exact match is tightly clustered across models, from 0.565 to 0.580, but the number of trajectories that survive the stricter baseline filter varies more. `DeepSeek-V3.1` yields 172 kept baselines, compared with 134 for `gpt-4.1` and 124 for `Llama-3.3-70B-Instruct`. The reason is simple: DeepSeek tends to make more tool calls, so more of its successful runs satisfy the "at least two searches" criterion.

The narrow spread in exact match is useful. It means the later replay results are not driven by a large clean-performance gap at the benchmark level. The more important difference appears after intervention, when the models must revise an already-started trajectory. Figure 10a in Appendix E.2 breaks the baseline collection down from total trajectories to exact-match runs to kept baselines.

Table 2. Full-baseline results. Kept baselines satisfy the criteria used for later intervention and replay.

Model	EM	F1	Kept	Mean steps
`gpt-4.1`	0.580	0.739	134	2.81
`DeepSeek-V3.1`	0.573	0.722	172	3.69
`Llama-3.3-70B-Instruct`	0.565	0.710	124	2.95

Filtering these 430 kept baselines yields 569 candidate intervention points, using the three-filter rule whose drop breakdown is detailed in Appendix E.2. To gauge how much the selected points matter, we replace a chosen tool output with an empty result and re-run; only 2 of 50 deletion replays fail, a 4% induced-failure rate. Missing evidence rarely breaks the agent, because the models often reformulate the query and recover. The harder case is fluent wrong evidence, which the remainder of the paper studies directly.

5.2. Main Counterfactual Replay Study

We generate 3,175 perturbations over the 569 intervention points. *WRONG_ATTRIBUTE* applies only when a supporting sentence contains a perturbable date, count, or relation, so it covers 336 points; the other perturbation families cover 567 to 569 points each. The counterfactual replay then resumes every perturbed run with honest later tool calls.

The main quantitative result is that cross-model variation is larger than cross-perturbation variation. Table 3 summarizes recovery and detection rates side by side. For `gpt-4.1`, every recovery cell is above 0.93. `DeepSeek-V3.1` sits between 0.876 and 0.930. `Llama-3.3-70B-Instruct` is consistently lower, with its worst performance on *PLAUSIBLE_IRRELEVANT* at 0.707. The table also makes the two-axis gap explicit: high-recovery cells often coincide with very low detection, most sharply for `gpt-4.1` on *SELF_CONTRADICTING* (0.979 recovery, 0.014 detection).

The most direct explanation is in the contradicting-evidence rate. When the post-intervention queries retrieve honest evidence that disagrees with the perturbation, recovery becomes much more likely. `gpt-4.1` sees contradicting evidence in roughly 90–95% of cells, compared with 81–88% for DeepSeek and 67–80% for Llama. The replay study therefore points to two linked mechanisms: some models are better at re-querying toward relevant evidence, and they are also better at using that evidence once they see it.

Figure 3 summarizes the main replay outcomes. The recovery plot shows the cross-model gap. The detection plot

Table 3. Recovery and detection rate by perturbation type and model. Recovery is exact match over all replay runs; detection is the LLM-judge noticing label (§3.5) over the judged runs (seven unparseable outputs excluded) and should be read as an upper bound. Per-cell n is the intervention-point count: 140/280–282/147 for the five full perturbations and 83/158/95 for *WRONG_ATTRIBUTE* (`gpt-4.1`/`DeepSeek`/`Llama`).

	`gpt-4.1`		`DeepSeek`		`Llama`	
Perturbation	Rec	Det	Rec	Det	Rec	Det
WRONG_ENTITY	0.964	0.108	0.900	0.312	0.776	0.265
WRONG_ATTRIBUTE	0.952	0.111	0.930	0.177	0.768	0.284
PLAUSIBLE_IRRELEVANT	0.971	0.950	0.876	0.968	0.707	0.932
EMPTY	0.936	0.250	0.883	0.709	0.762	0.524
SELF_CONTRADICTING	0.979	0.014	0.907	0.136	0.776	0.103
AUTHORITATIVE_WRONG	0.964	0.093	0.911	0.276	0.735	0.204
Mean	0.961	0.254	0.901	0.430	0.754	0.385

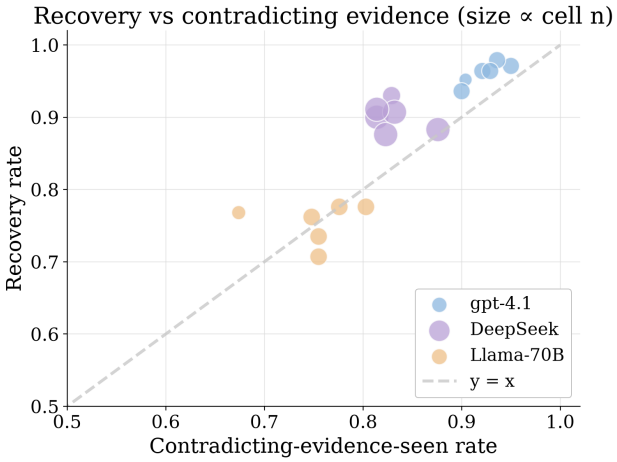


Figure 2. Recovery rate against contradicting-evidence-seen rate for each model-by-perturbation cell. Cells with higher exposure to contradicting evidence also recover more often, which makes later evidence acquisition the main upstream determinant of recovery.

shows a different pattern. Obvious perturbations such as *PLAUSIBLE_IRRELEVANT* and *EMPTY* trigger explicit noticing. Fluent wrong paragraphs often do not.

5.3. Recovery Is Common, but Usually Silent

We code the 3,175 replay trajectories into four outcome classes. Seven judge outputs fail to parse, leaving 3,168 valid trajectories. Across those runs, 54.8% are *SILENTLY_CORRECTED*, 32.7% are *NOTICED_AND_RECOVERED*, 6.2% are *NOTICED_BUT_FAILED*, and 6.2% are *PERSISTED*.

This is the central empirical point of the paper. Final correctness alone suggests that the agents are robust in most runs. Outcome coding shows that the mechanism of robustness is often silent. The model reaches the right answer without stating that a previous observation was inconsistent. This matters for evaluation because trace inspection can underestimate recovery, and it also matters for interpretability because explicit reasoning is not a reliable record of belief revision.

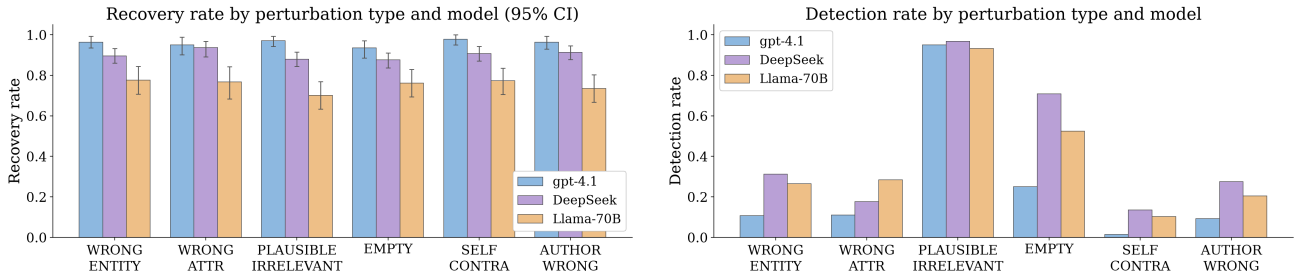
Aggregated across models and perturbation types (Figure 3), the most stable pattern is that *PLAUSIBLE_IRRELEVANT* is almost always noticed, while *SELF_CONTRADICTING* is rarely noticed. That latter result is stronger than we expected. Even when the paragraph contains both a fact and its contradiction, the agent often does not say that the two conflict.

5.4. Follow-up Analyses

The remaining analyses refine the interpretation rather than changing it. We organize them by what they add to the main result: a prompt-level mitigation, a cross-dataset replication, pairwise significance tests, qualitative failure subtypes, and a judge robustness check.

Cross-check intervention. We add a single instruction to the system prompt asking the agent, before giving the final answer, to review whether any earlier tool output may have been wrong or contradicted by later evidence, and to prefer the later evidence (the exact suffix appended to the baseline system prompt is reproduced in Appendix B.3). Replaying all 3,175 perturbations under this prompt shifts the overall outcome distribution. *PERSISTED* falls from 6.2% to 3.5%, a 44% relative reduction. *NOTICED_AND_RECOVERED* rises from 32.7% to 42.6%. *SILENTLY_CORRECTED* falls from 54.8% to 48.7%, consistent with mass moving from silent to explicit recovery. The effect is strongly model-dependent. `gpt-4.1`'s average recovery is essentially unchanged at 0.961 versus 0.953, within the bootstrap interval. `DeepSeek-V3.1`'s rises from 0.901 to 0.920. `Llama-3.3-70B-Instruct`'s rises from 0.754 to 0.872, with per-perturbation gains ranging from 7 to 14 percentage points. The instruction adds roughly 80 prompt tokens per call and produces its largest absolute gain on the model that started furthest from ceiling. Figure 4 shows the overall shift.

Cross-dataset replication. We next test whether the cross-model ordering holds on a second multi-hop benchmark with longer reasoning chains. We sample 100 MuSiQue-Ans (Trivedi et al., 2021) development questions for each of 2, 3, and 4 hops, translate each into the same context schema as HotpotQA, and run the same harness across the three models. The first phase collects 852 successful baseline trajectories out of 900 attempted; the 48 lost baselines fall almost entirely on `DeepSeek-V3.1` (254 usable versus 298 for `gpt-4.1` and 300 for `Llama-3.3-70B-Instruct`), reflecting its higher rate of transient harness errors. Per-hop baseline exact match falls in the expected direction. `gpt-4.1` drops from 0.57 at 2-hop to 0.37 at 3-hop and 0.40 at 4-hop. `DeepSeek-V3.1` follows the same shape (0.54, 0.34, 0.37). `Llama-3.3-70B-Instruct` sits be-



(a) Recovery rate *(b)* Detection rate

Figure 3. Main outcomes from the perturbation study. Recovery varies mostly by model. Explicit detection varies strongly by perturbation type.

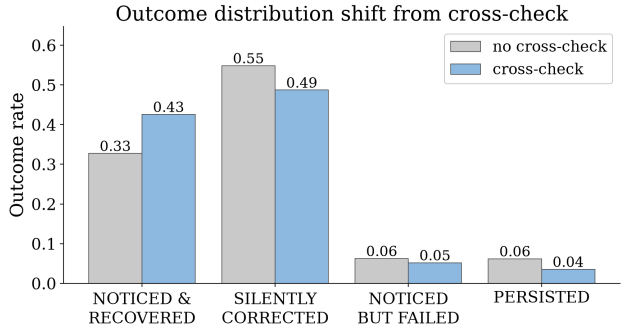


Figure 4. Outcome distribution before and after the cross-check intervention. The instruction moves mass from *SILENTLY_CORRECTED* to *NOTICED_AND_RECOVERED* and reduces *PERSISTED* by roughly 44% in relative terms.

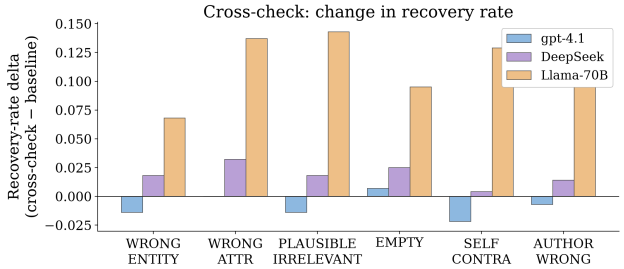


Figure 5. Per-model, per-perturbation change in recovery rate from the cross-check intervention. `Llama-3.3-70B-Instruct` gains 7–14 percentage points on every perturbation type. `DeepSeek-V3.1` gains a few points across most types. `gpt-4.1` shows essentially no change, with small negative deltas inside bootstrap noise. The before-and-after curves themselves are reproduced in Appendix D.

Table 4. Cross-dataset *WRONG_ENTITY* recovery. The HotpotQA column repeats the *WRONG_ENTITY* row of Table 3; MuSiQue is broken out by hop and weighted across hops by cell size. Per-hop n ranges from 20 to 47 (285 runs total).

	HotpotQA	MuSiQue by hop			MuSiQue
Model	(WE)	2-hop	3-hop	4-hop	(wt.)
`gpt-4.1`	0.964	0.936	0.938	1.000	0.956
`DeepSeek-V3.1`	0.900	0.909	0.875	0.826	0.879
`Llama-3.3-70B-Instruct`	0.776	0.973	0.875	0.900	0.926

low both at every hop count (0.44, 0.28, 0.25). The cross-model ordering matches HotpotQA at every hop count. Appendix C reports per-hop baseline exact match as a separate figure.

The replay study uses *WRONG_ENTITY* only, applied at the first-hop tool call. After filtering for clean baselines and clean intervention points, we obtain 285 perturbed runs across the nine (model, hop) cells, with per-cell sample sizes ranging from 20 to 47. Figure 6 shows recovery rate by hop count, and Figure 7 compares the weighted aggregate against HotpotQA; Table 4 lists the exact per-hop and weighted rates.

Three patterns are visible in the replay results. First, the cross-model ordering at the *WRONG_ENTITY* recovery level matches HotpotQA. Weighted across hops, recovery is 0.96 for `gpt-4.1`, 0.88 for `DeepSeek-V3.1`, and 0.93 for `Llama-3.3-70B-Instruct`. The first two values agree with their HotpotQA counterparts within about two percentage points. Second, the per-hop trajectory differs by model. `gpt-4.1`'s recovery climbs in hop count, from 0.94 at 2-hop to 0.94 at 3-hop to 1.00 at 4-hop. `DeepSeek-V3.1`'s declines monotonically (0.91, 0.88, 0.83), with catastrophic-failure rate roughly doubling from 9.1% to 17.4% over the same range. `Llama-3.3-70B-Instruct`'s is non-monotonic, peaking at 0.97 for 2-hop, dropping to 0.88 at 3-hop, then partially recovering at 4-hop. Third, mean post-intervention tool calls generally grows with hop count, most steeply for `DeepSeek-V3.1` (2.73, 3.67, 5.09 across the three hop counts); `Llama-3.3-70B-Instruct` is the exception, leveling off after 3-hop (1.49, 2.67, 2.30). Appendix C reports the verification-effort figure.

The `Llama-3.3-70B-Instruct` improvement deserves an explanation. Its weighted MuSiQue recovery of 0.93 is meaningfully higher than its HotpotQA value of 0.78. Two factors plausibly contribute. The first is the larger paragraph pool. MuSiQue gives the agent 20 paragraphs

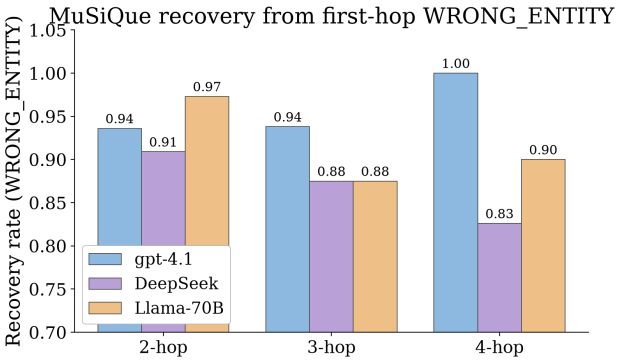


Figure 6. MuSiQue recovery rate from a first-hop *WRONG_ENTITY* perturbation, broken out by model and hop count. `gpt-4.1` climbs to 1.00 at 4-hop. `DeepSeek-V3.1` declines monotonically. `Llama-3.3-70B-Instruct` is non-monotonic but stays in the 0.88–0.97 band.

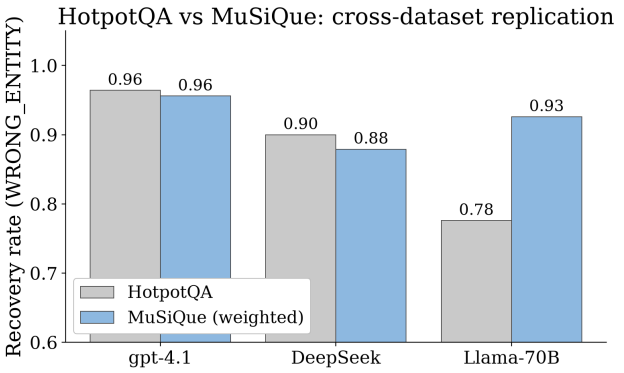


Figure 7. Cross-dataset comparison of *WRONG_ENTITY* recovery on HotpotQA and MuSiQue (weighted across 2, 3, and 4-hop chains). The cross-model ordering replicates, and absolute rates agree closely for the two stronger models.

per question instead of 10, which raises the chance that a reformulated query lands on a paragraph containing the unperturbed answer. The second is the choice to intervene only at the first hop. With at least one and often two more honest hops to traverse, every *WRONG_ENTITY* replay leaves multiple opportunities for the corruption to be displaced. Both factors apply less strongly in the original HotpotQA setup, where the second hop is often the last step and the intervention can therefore land later in the chain.

Pairwise significance. We run pairwise permutation tests across the six perturbation types within each model, with Holm correction over the fifteen pairs. After correction, no recovery-rate comparison within any model is significant (all $p_{\text{adj}} \geq 0.70$, and exactly 1.0 for every pair except two on `DeepSeek-V3.1`). Detection-rate comparisons are different. In every model, *PLAUSIBLE_IRRELEVANT* differs significantly from each of the other five types ($p_{\text{adj}} = 0.003$), and *SELF_CONTRADICTING* differs from both *EMPTY* and *WRONG_ENTITY* (for `gpt-4.1`, $p_{\text{adj}} = 0.003$ and 0.007 respectively). The clean phrasing of this finding is that the model factor dominates the perturbation factor for whether the run recovers, while perturbation type is the main driver of whether the agent verbalizes the conflict.

Qualitative failure subtypes. We sample 20 trajectories from each outcome class, stratified across model and perturbation type. The qualitative read identifies four named failure subtypes within the wrong-answer classes. Inside *PERSISTED*, three patterns recur: *authoritative cooption*, where the confidence prefix in *AUTHORITATIVE_WRONG* suppresses further verification; *self-contradiction collapse*, where the agent treats the inserted contradicting sentence as a second fact rather than recognizing the conflict; and *premature termination*, more common on `Llama-3.3-70B-Instruct`, where the agent commits to an answer after the first or second search and never sees later honest evidence. Inside *NOTICED_BUT_FAILED*, the dominant pattern is *conflict-anchored mistake*: the agent verbalizes the contradiction explicitly, then anchors on the perturbed first observation rather than the later honest one. In both *NOTICED_AND_RECOVERED* and *SILENTLY_CORRECTED*, 19 of 20 sampled runs saw contradicting evidence. Failure classes also often saw contradicting evidence, so additional verification effort by itself is not enough.

Alternate-judge robustness. We re-judge 300 trajectories sampled across (model, perturbation type, outcome) using `DeepSeek-V3.1` in place of `gpt-4.1`. Inter-judge agreement is 78.7%. The 64 disagreements are sharply asymmetric: in 61 of them, the `gpt-4.1` judge calls the trajectory "detected" and `DeepSeek-V3.1` calls it "not detected"; only 3 cases run the other direction. The cross-perturbation ordering survives the judge swap, with *PLAUSIBLE_IRRELEVANT* remaining the most-noticed and *SELF_CONTRADICTING* the least. The absolute detection rates reported earlier in this section should therefore be read as an upper bound. Recovery rates and the four outcome counts that depend only on correctness are unaffected by the judge change.

6. Conclusion

Counterfactual replay exposes a specific failure mode that final-answer evaluation misses. Tool-using agents often recover from corrupted intermediate evidence, but they usually do so silently. The stronger the model, the more often the run recovers, yet explicit contradiction detection remains sparse for fluent wrong evidence. The main practical implication is that final success is not enough to understand trajectory robustness. Evaluating belief revision requires interventions inside the run, not just scoring the endpoint.

7. Limitations

The study stays within HotpotQA's distractor setting. That gives us the control needed for counterfactual replay, but it also limits the tool surface to retrieval over a fixed paragraph pool. The explicit-detection labels are judge-dependent, as our alternate-judge check shows. The intervention-point filter also favors trajectories where supporting facts are easy to align with later mentions, which is useful for precision but may leave out harder forms of propagation.

The study also does not yet test broader interactive agent benchmarks with richer tool surfaces or simulated users. The qualitative appendix helps interpret the labels, but it is still a small sample relative to the full replay set.

References

Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems 37*, NeurIPS 2024, pp. 82895–82920. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-2636. URL http://dx.doi.org/10.52202/079017-2636.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *AISec@CCS*, 2023. doi: 10.1145/3605764.3623985.

Hou, Y., Pascale, A., Carnerero-Cano, J., Tchrakian, T., Marinescu, R., Daly, E., Padhi, I., and Sattigeri, P. Wikicontradict: A benchmark for evaluating llms on real-world knowledge conflicts from wikipedia. In *Neural Information Processing Systems (NeurIPS)*, 2024. doi: 10.52202/079017-3481.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C. E., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., Lukovsiut.e, K., Nguyen, K., Cheng, N., Joseph, N., Schiefer, N., Rausch, O., Larson, R., McCandlish, S., Kundu, S., Kadavath, S., Yang, S., Henighan, T., Maxwell, T. D., Telleen-Lawton, T., Hume, T., Hatfield-Dodds, Z., Kaplan, J., Brauner, J., Bowman, S., and Perez, E. Measuring faithfulness in chain-of-thought reasoning. *ArXiv*, abs/2307.13702, 2023.

Lee, J., Lee, K., and Kim, T. Magic: A multi-hop and graph-based benchmark for inter-context conflicts in retrieval-augmented generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., and Scialom, T. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*, 2024.

Paul, D., West, R., Bosselut, A., and Faltings, B. Making reasoning matter: Measuring and improving faithfulness of chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15012–15032. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.882. URL http://dx.doi.org/10.18653/v1/2024.findings-emnlp.882.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36*, NeurIPS 2023, pp. 68539–68551. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/075280-2997. URL http://dx.doi.org/10.52202/075280-2997.

Sharma, M., Tong, M., Korbak, T., Duvenaud, D., Askell, A., Bowman, S. R., Cheng, N., Durmus, E., Hatfield-Dodds, Z., Johnston, S., Kravec, S., Maxwell, T., McCandlish, S., Ndousse, K., Rausch, O., Schiefer, N., Yan, D., Zhang, M., and Perez, E. Towards understanding sycophancy in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Sun, J., Min, S. Y., Chang, Y., and Bisk, Y. Tools fail: Detecting silent errors in faulty tools. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14272–14289, 2024.

Sun, K., Bai, F., and Dredze, M. Task matters: Knowledge requirements shape llm responses to context–memory conflict. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. doi: 10.18653/v1/2026.findings-acl.202.

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 10:539–554, 2021. doi: 10.1162/tacl_a_00475.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Neural Information Processing Systems (NeurIPS)*, 2023.

Vuddanti, S. V., Shah, A., Chittiprolu, S. K., Song, T., Dev, S., Zhu, K., and Chaudhary, M. Paladin: Self-correcting language model agents to cure tool-failure cases. *ArXiv*, abs/2509.25238, 2025.

Wang, R., Chen, Y., Wang, Y., Wu, C., Fang, J., Cai, X., Gu, Q., Su, H., Zhang, A., Wang, X., Cai, X., and Chua, T.-S. AgentNoiseBench: Benchmarking robustness of tool-using LLM agents under noisy condition. In *International Conference on Machine Learning (ICML)*, 2026.

Wei, J. W., Huang, D., Lu, Y., Zhou, D., and Le, Q. V. Simple synthetic data reduces sycophancy in large language models. *ArXiv*, abs/2308.03958, 2023.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045, 2024.

Zeng, L., Gupta, R., Motwani, D., Yang, D., and Zhang, Y. Worse than zero-shot? a fact-checking dataset for evaluating the robustness of rag against misleading retrievals. In *Neural Information Processing Systems (NeurIPS)*, 2025.

Zhan, Q., Liang, Z., Ying, Z., and Kang, D. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 10471–10506. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL http://dx.doi.org/10.18653/v1/2024.findings-acl.624.

Zhang, G.-M., Wang, J., Chen, J., Zhou, W., Wang, K., and Yan, S. Agentracer: Who is inducing failure in the llm agentic systems? *ArXiv*, abs/2509.03312, 2025a.

Zhang, S., Yin, M., Zhang, J., Liu, J., Han, Z., Zhang, J., Li, B., Wang, C., Wang, H., Chen, Y., and Wu, Q. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *International Conference on Machine Learning (ICML)*, 2025b.

Zhu, K., Liu, Z., Li, B., Tian, M., Yang, Y., Zhang, J., Han, P., Xie, Q., Cui, F., Zhang, W., Ma, X., Yu, X., Ramesh, G., Wu, J., Liu, Z., Lu, P., Zou, J., and You, J. Where llm agents fail and how they can learn from failures. *ArXiv*, abs/2509.25370, 2025.

Zou, W., Geng, R., Wang, B., and Jia, J. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *USENIX Security Symposium*, pp. 3827–3844, 2024.

A. Qualitative Trajectory Examples

This appendix gives one representative sample for each outcome class from the qualitative audit. The goal is not to reproduce full traces, but to show the behavioral signature that motivated the coding scheme in Table 1.

NOTICED_AND_RECOVERED. DeepSeek-V3.1, *SELF_CONTRADICTING*.
Question. Which singer-songwriter achieved his third No. 1 single by writing "Always and Forever"?
Perturbation. The first retrieved paragraph mixed the true song title "Forever and Ever, Amen" with contradictory inserted variants.
Trace signal. After the intervention, the agent explicitly says: "There might be confusion or a different song. I should check if 'Always and Forever' is a separate song or if it's a misstatement." It then issues targeted follow-up searches for `Paul Overstreet No 1 singles` and `Always and Forever country No 1 single`.
Outcome. The agent returns *Paul Overstreet*. This is the clearest repair pattern in the dataset: explicit mismatch recognition followed by focused retrieval aimed at resolving the contradiction.

SILENTLY_CORRECTED. Llama-3.3-70B-Instruct, *WRONG_ENTITY*.
Question. Walt Williams attended which institution in 1988, and what conference is it in today?
Perturbation. The first observation corrupted the player name, yielding "Roger Curtis 'The Wizard' Williams" in the paragraph about Walt Williams.
Trace signal. The agent never says that the name is wrong. Instead, it extracts the unaffected structural fact that the player attended the University of Maryland, immediately queries `University of Maryland conference`, and answers from that second path.
Outcome. The final answer is *Big Ten Conference*. Recovery happens, but there is no explicit contradiction statement. This is why recovery and noticing are separated in our coding.

NOTICED_BUT_FAILED. gpt-4.1, *EMPTY*.
Question. Maggie Gripenberg introduced a teaching method to Finland; what is that method used to teach?
Perturbation. The first tool call returned no result.
Trace signal. The agent explicitly notes the missing evidence: "Since there was no information about Maggie Gripenberg's teaching method, I should search for Maggie Gripenberg's profession or area of expertise." It then searches for her profession, sees that she was a pioneer of modern dance, and infers the answer from that nearby but incomplete cue.
Outcome. The final answer is *movement and dance*, while the gold answer is *music*. The agent noticed the failure, searched again, and still resolved the conflict incorrectly by overgeneralizing from adjacent evidence.

PERSISTED. Llama-3.3-70B-Instruct, *EMPTY*.
Question. Ladislav Matejka's Michigan Slavic Publications published volumes by which historian considered to be the founder of morphophonology?
Perturbation. The initial retrieval was empty.
Trace signal. Unlike the previous case, the agent never marks the missing retrieval as a problem. It jumps to a prior expectation about Prague School linguistics, searches for `Roman Jakobson Michigan Slavic Publications`, finds that Jakobson was one of the published authors, and stops there.
Outcome. The final answer is *Roman Jakobson*, but the gold answer is *Nikolai Trubetzkoy*. This is a persistence failure: the trajectory contains no explicit detection signal, and the gap left by the empty tool output is filled by a plausible but wrong latent hypothesis.

B. Prompt Templates Used in the Study

This appendix records the exact LLM prompt templates used in the study. The alternate-judge robustness check uses the same judge prompt as the main coding pass, but swaps the judge model from `gpt-4.1` to `DeepSeek-V3.1`. The cross-check intervention uses the baseline ReAct system prompt plus the suffix shown below.

B.1. Baseline ReAct System Prompt

```
You are a research-style question-answering agent with access to one tool:
  search(query: str) -> a list of up to 2 Wikipedia-style paragraphs

Use search to gather facts before answering. Each search returns up to two paragraphs
from a fixed corpus, formatted as:
  Title: <paragraph title>
  [1] <sentence 1>
  [2] <sentence 2>
  ...

At every step, write exactly one of these two response forms, and nothing else:

  Thought: <your reasoning>
  Action: search("<your query>")

OR

  Thought: <your reasoning>
  Final Answer: <short answer; an entity name, a number, or "yes"/"no">

Rules:
  - Always begin your response with "Thought:".
  - Do not write "Observation:" yourself; the user will append observations after each
Action.
  - Keep the final answer short (one entity, one number, or yes/no). Do not include
surrounding prose.
  - You have a maximum of {max_steps} steps in total.
```

B.2. Runtime User Message Wrapper

```
Initial user turn:
Question: <question text>

Tool feedback turn appended after each action:
Observation: Title: <paragraph title>
[1] <sentence 1>
[2] <sentence 2>
...
```

B.3. Cross-Check Intervention Suffix

```
CROSS-CHECK INSTRUCTION:
Before producing your Final Answer, briefly review whether any of your earlier tool
outputs may have been wrong or contradicted by later evidence. If so, prefer the later
evidence and state which earlier output you are choosing to override.
```

B.4. Judge Prompt Template

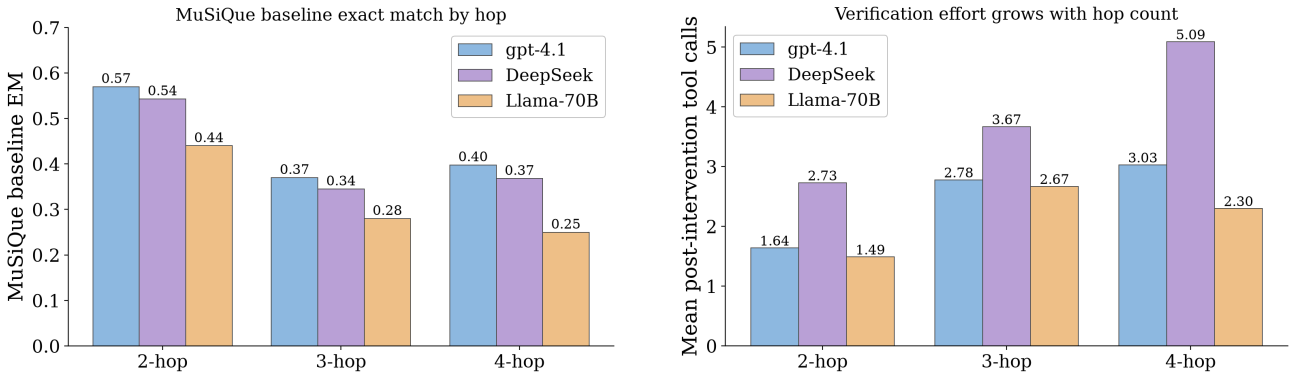
```
You will be shown the post-intervention reasoning trace of a question-answering agent.
The agent had access to a search tool.  At one earlier step, the tool returned a
wrong/manipulated paragraph (the "perturbation").  Your job is to read the agent's
later thoughts, actions, and final answer, and decide whether the agent EXPLICITLY
NOTICED that something was wrong with an earlier tool output.

Detection signals you should treat as YES:
  - Explicit acknowledgement of inconsistency, contradiction, or doubt about earlier
evidence
    (e.g.  "the previous result said X but this says Y", "I am unsure whether...", "Let
me re-check...")
  - Re-querying a fact that was already returned earlier, with language indicating
verification
  - Re-reading or re-evaluating earlier tool outputs

Do NOT count as detection:
  - Mere additional searches without expressed doubt
  - Routine continuation of the trajectory
  - The agent simply giving a different final answer than the perturbation suggested

Reply with strict JSON only, no surrounding prose:
{"detected":  true|false, "evidence":  "<short quoted phrase from the trace, or '' if
not detected>"}

--- TRACE BEGINS ---
QUESTION: %s
GOLD ANSWER: %s
PERTURBATION TYPE: %s
PERTURBATION META: %s

POST-INTERVENTION TRACE (steps after the corrupted tool output):
%s

FINAL ANSWER GIVEN BY AGENT: %s
--- TRACE ENDS ---
```

C. MuSiQue Generalization: Supplementary Figures

This appendix gives two MuSiQue figures that are referenced in Section 5 but not reproduced inline. Per-hop baseline exact match (Figure 8a) gives the clean-task difficulty profile that the replay study sits on top of. Per-hop verification effort (Figure 8b) gives the post-intervention tool-call counts that motivate the observation that `DeepSeek-V3.1` keeps searching when uncertain. Table 5 lists the full per-hop numbers behind these figures.

Table 5. MuSiQue replay detail, per model and hop count (*WRONG_ENTITY* injected at the first hop). Baseline EM is the clean-run exact match; recovery, catastrophic rate, and mean post-intervention tool calls are over the perturbed runs.

Model	Hop	n	Base EM	Recovery	Catastr.	Post-tools
`gpt-4.1`	2	47	0.570	0.936	0.064	1.64
	3	32	0.370	0.938	0.062	2.78
	4	34	0.398	1.000	0.000	3.03
`DeepSeek-V3.1`	2	44	0.543	0.909	0.091	2.73
	3	24	0.345	0.875	0.125	3.67
	4	23	0.368	0.826	0.174	5.09
`Llama-3.3-70B-Instruct`	2	37	0.440	0.973	0.027	1.49
	3	24	0.280	0.875	0.125	2.67
	4	20	0.250	0.900	0.100	2.30



(a) Baseline exact match by hop

(b) Verification effort by hop

Figure 8. MuSiQue generalization. (a) Baseline exact match by hop count on the 100-question per-hop sample; the cross-model ordering matches HotpotQA at every hop count, and exact match drops sharply from 2-hop to 3-hop with a smaller change between 3-hop and 4-hop. (b) Mean post-intervention tool calls by model and hop count; `DeepSeek-V3.1` scales most steeply (2.73, 3.67, 5.09 across 2/3/4-hop), while `Llama-3.3-70B-Instruct` shows little additional verification effort beyond 3-hop, consistent with its tendency to commit early.

D. Cross-Check Intervention: Supplementary Figure

This appendix gives the per-cell before-and-after recovery curves for the cross-check intervention. Solid lines are recovery rates with the cross-check instruction added, dashed lines are the same cells without it. The same per-model story is visible: the Llama curves shift up the most, DeepSeek shifts up a little, and the gpt-4.1 curves are essentially indistinguishable from each other.

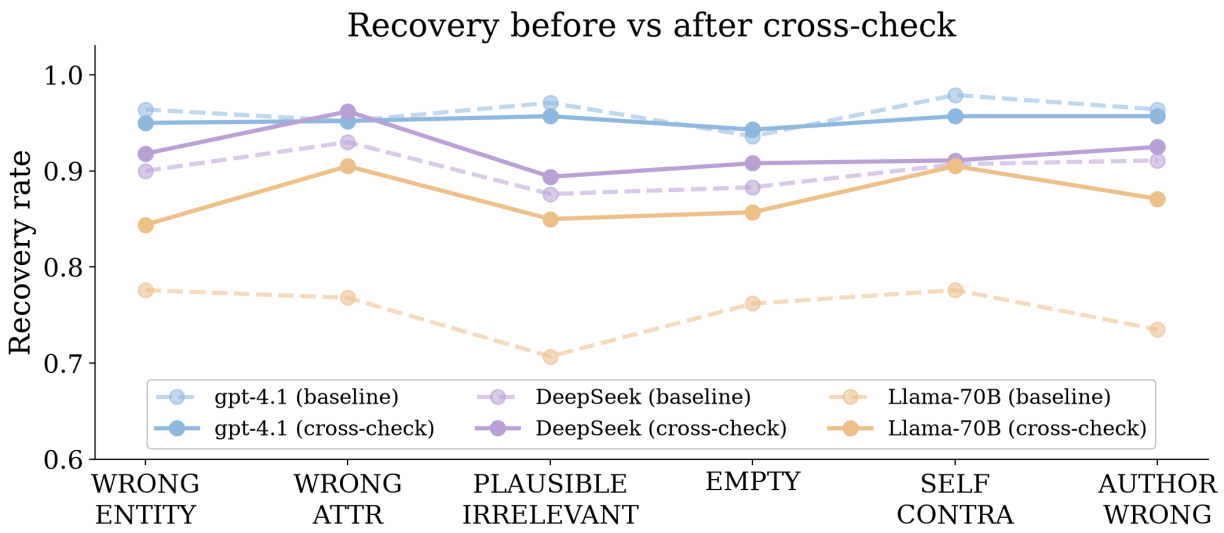


Figure 9. Recovery rate per (model, perturbation type) cell, before (dashed) and after (solid) the cross-check intervention. The vertical separation between dashed and solid lines is largest for `Llama-3.3-70B-Instruct` and smallest for `gpt-4.1`, consistent with the per-model deltas in Figure 5.

E. Dataset and Baseline Statistics

E.1. Dataset Funnel and Study Volume

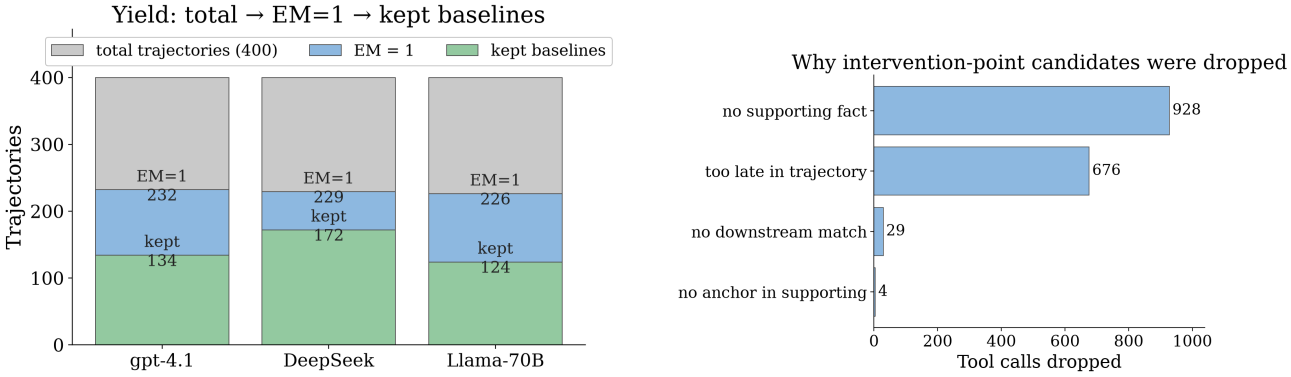
Table 6 records the full study funnel, from the raw HotpotQA distractor split through to the judged perturbed runs used in the main analysis.

Table 6. Dataset funnel and study volume. The judged total excludes 7 trajectories whose judge output did not parse as JSON.

Stage	Count
HotpotQA dev distractor split	7,405
Filtered pool	5,370
Candidate set	400
Baseline trajectories (3 models)	1,200
Kept baselines	430
Intervention points	569
Generated perturbations	3,175
Judged perturbed runs	3,168

E.2. Baseline Yield and Intervention-Point Filtering

Figure 10a breaks the baseline collection down from the full set of trajectories to the exact-match runs and finally to the kept baselines that satisfy the structural criteria required for intervention. Figure 10b then shows why candidate tool calls are dropped when selecting intervention points: the largest filter loss comes from tool outputs that do not contain supporting facts; the position constraint, which requires at least two later steps, is the second largest; and the downstream-reference filter removes very little, which means that once a retrieved paragraph contains a supporting fact, the agent usually uses it later.



(a) Baseline yield

(b) Intervention-point filtering

Figure 10. Baseline yield and intervention-point filtering. (a) Yield from total trajectories to exact-match runs to kept baselines; the main bottleneck for downstream perturbation is not clean accuracy alone, but how many successful runs also satisfy the structural criteria required for intervention. (b) Why intervention-point candidates were dropped: most candidate tool calls fail because they do not contain a supporting fact, and a second large share occurs too late in the trajectory to permit later contradiction.