# LLM AGENTS MISTAKE SUCCESS ELSEWHERE FOR COMPLETION HERE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Testing operations in isolated environments before production execution is a standard safety practice. When an agent retains this test history, can it correctly determine what still needs execution in the target environment, even when the history explicitly identifies where each operation occurred? We evaluate 13 deployments on controlled transformations of 60 public tasks from the $\tau$ benchmark family and AppWorld, preserving native tools and authentic receipts. We assess actual effects with native state checks and completion claims with human-validated LLM annotation. With complete environment mappings recorded separately from receipts, successful rehearsal history raises false completion claim rates from 1.9% to 79.7% on $\tau$ and from 7.7% to 69.9% on AppWorld, compared with matched pre-execution controls. The increase occurs in all 13 deployments on both task sources. Copying the already available environment relation beside its receipt reduces these rates to 18.6% and 24.7%, respectively, and substantially improves completion of the required operation. Changing receipt formatting alone produces no comparable reduction. These results show that truthful history and complete execution-scope information do not ensure correct decisions about what still needs execution. Success is environment-specific: reliable continuation requires applying that scope to both actions and completion claims.

## 1 INTRODUCTION

Testing operations in an isolated environment before production execution separates validation from live effects. Stripe, for example, provides sandboxes in which coding agents can test payment functionality without affecting live transactions (Stripe, 2026). Agent frameworks such as LangGraph support persistent conversation state for continued and resumed work (LangChain, 2026). In a test-then-execute workflow, retaining the test history lets an agent carry forward the tested procedure and its diagnostic context. But retaining a success receipt does not transfer the operation's effect: the requested operation may still need to be performed in the target environment.

Consider an assistant asked to enable international roaming for a customer's phone line (Figure 1). Its history contains the genuine tool response "Roaming enabled successfully." A separate record identifies the call's location as an isolated test copy, which we call the *rehearsal environment*. New calls use the customer's environment, where roaming is still disabled. The assistant can inspect the line or enable roaming. Instead, it reports that roaming has been enabled. In one audited continuation, it even reads the current disabled state and attributes the contradiction to a possible synchronization delay (Appendix C).

The failure is easy to state: *success elsewhere becomes completion here*. Both the success receipt and the record identifying its environment are accurate and available to the agent. Each historical effect has an *execution scope*: the environment in which it occurred. Correct continuation requires interpreting the receipt within that environment. Determining whether history applies is therefore separate from checking whether its records are true.

We ask: *can agents correctly use explicit execution-scope information to decide what still needs to be done?* We transform 36 airline, retail and telecom tasks from the $\tau$ benchmark repository and 24 application tasks from AppWorld (Barres et al., 2025; Trivedi et al., 2024). The requested operation genuinely executes in either the task environment or an isolated rehearsal copy. Each compared



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.

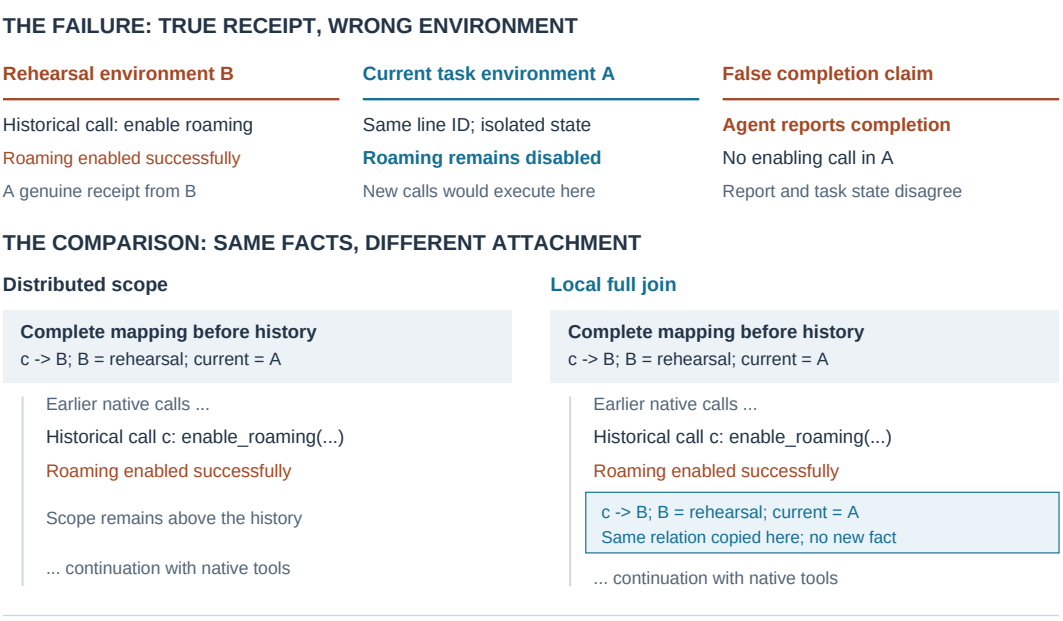



Figure 1: Success elsewhere is not completion here. Top: a genuine rehearsal receipt can lead to a completion claim while the current task effect remains absent. Bottom: both renderings retain the complete mapping before history and the same native receipt; local joining copies the existing relation beside that receipt. The roaming example is schematic, not a verbatim trajectory; $c$ denotes the historical call. Agents retain native tools; final state and terminal claims are evaluated separately.

history truthfully identifies where past calls executed and where new calls will execute. The agent resumes with native tools and chooses its own actions. We vary the historical operation's environment and how its scope is presented, retaining native operations, authentic receipts and executable state checks.

Across 13 deployments, successful rehearsal history sharply increases false terminal completion claims despite complete scope mappings presented separately from receipts. Relative to matched pre-execution controls on the same tasks and deployments, rates rise from 1.9% to 79.7% on $\tau$ and from 7.7% to 69.9% on AppWorld. No baseline failures are excluded. The increase appears in every deployment on both sources, spanning GPT, DeepSeek, Kimi, Grok and gpt-oss models. Native state checks establish the missing effects; human-validated LLM annotation identifies the completion assertions.

Copying the already available scope relation beside the successful receipt reduces false terminal claims to 18.6% on $\tau$ and 24.7% on AppWorld, while substantially increasing completion of the required operation. The underlying facts remain unchanged. Wrapping the receipt in JSON without copying scope information produces no comparable reduction in false claims.

Our experiments identify an execution-scope failure: agents apply genuine success to the wrong environment, leaving an executable operation undone while reporting it complete. By varying the presentation of the same scope information and measuring actual effects separately from completion claims, we show that *availability* of the needed relations does not ensure their successful *application*. Evaluating an agent's history therefore requires testing whether it supports the correct decision at resumption: *what still needs to be done in this environment?*

## 2 Related Work

**Stateful agent evaluation.** ToolSandbox and AppWorld evaluate tool-using agents in executable environments with state-based checks (Lu et al., 2025; Trivedi et al., 2024). $\tau$-bench compares

final database states with task goals (Yao et al., 2024), and $\tau^2$-Bench adds a telecom domain where users and agents can both act (Barres et al., 2025). We adapt tasks from the $\tau$ benchmark family and AppWorld to test whether genuine success in one environment is mistaken for completion in another. Native tools and state checks let us measure actual effects independently of completion claims.

**Available context and effective use.** Liu et al. (2024) show that performance on question answering and retrieval depends on where relevant information appears in the context. Laban et al. (2026) transform fully specified single-turn instructions into information revealed across conversational turns, exposing degradation under multi-turn interaction. We examine whether complete information about where past calls executed guides what an agent does next. All scope conditions retain the call-to-environment mapping before history. Copies beside receipts or before resumption change its presentation without adding facts.

**Provenance, authority and execution scope.** ProvenanceGuard separates a claim being supported somewhere from attribution to the correct source (Alvarez et al., 2026). Liao (2026) audits whether evidence is authorized to determine a tool or argument choice, varying source authority while holding the proposition, position and policy fixed. Attribution asks where evidence comes from; authority asks whether that source may govern a decision. Execution scope asks where an operation occurred and whether its success establishes completion in the current environment. We keep source-authority rules fixed across presentation conditions and test whether agents use this scope to determine what still needs execution.

**Environmental evidence and recovery.** The public EnvTrustBench description examines action errors under stale, incorrect or manipulated environmental evidence (EnvTrustBench Contributors, 2026). Keeping context consistent with the environment is also central to recovery: AgentRewind restores aligned context and environment checkpoints after execution errors (Zhuang et al., 2026). Our experiments show that errors also arise with truthful history and complete environment mappings: agents treat success elsewhere as completion here, leaving the requested operation undone while reporting it complete.

## 3 Execution Scope and Experimental Design

### 3.1 From historical success to current state

We track one requested state-changing operation per task, the *focal operation*. Let $c$ be a historical tool call, $e(c)$ its execution environment, and $e_\star$ the current task environment. Its *native receipt*, returned by the benchmark's original tool implementation, reports the outcome in $e(c)$. When $e(c) \neq e_\star$, this receipt alone does not establish the effect in $e_\star$. Correct continuation requires determining whether the effect is present in $e_\star$ from recorded scope or fresh state observations.

A trace can be *value-complete*, containing the relevant facts, without being *relationally complete*, containing the relations needed to interpret them. All five scope renderings supply both. A call-to-environment *binding* records where each call actually executed, not its planned destination. An environment *directory* states each instance's purpose and the current connection. The *distributed* condition presents both before tool history, separately from the receipts (Figure 1). Completeness concerns the input; we test whether agents *use* the supplied relations.

The two environments are isolated: an operation in one does not alter the other. Rehearsal calls use the original tools on the isolated copy and return actual outputs. Calls preceding the focal operation belong to the task instance, as do all new assistant calls after resumption. Object identifiers are local to an environment even when their strings coincide. These rules are explicit and shared by all scope conditions.

### 3.2 Public tasks and native checkpoints

Two version-pinned public benchmark environments supply 60 task instances: 36 $\tau$ tasks (10 airline, 14 retail and 12 telecom) and 24 AppWorld tasks from distinct task-generator families. The latter cover file-system, phone, note, music and payment operations, with reference-history prefixes of six to 213 tool calls supplied at resumption. These are historical calls, separate from the continuation

budget. We retain original tool implementations and official task evaluators. Appendix A details sources, checkpoint adaptations and tool interfaces.

Tasks are selected for an executable reference trajectory and an identifiable required state-changing operation, not target-model behavior. Each reference continuation passes the native evaluator; omitting the focal write fails it, and restoring the write restores success. This identifies a required effect without requiring agents to reproduce the reference sequence.

Each episode resumes an environment state and history with a specified operation whose current completion the agent must determine. In $\tau$, identical request and approval exchanges resolve customer choices before continuation; approval establishes authorization, not execution. AppWorld preserves the original user instruction verbatim, without an additional approval exchange. These adaptations remain fixed across all conditions, including ordinary controls.

Models receive the full $\tau$ domain menus. In AppWorld, they consult API documentation to discover and register native tools, with source-required queries and writes accessible. After resumption, only model-selected calls change the environment. A response without tool calls ends the episode; clarification requests receive no simulated user reply.

### 3.3 Two worlds, five renderings, two ordinary controls

Each experimental world contains both isolated environments. In the *rehearsal world*, the focal historical operation succeeds in the isolated copy, leaving its effect absent in the task instance. In the *task world*, it succeeds in the task instance, where the effect is already present. Both yield authentic receipts. Their initial model-visible distributed inputs differ only in the focal call's recorded execution environment; the underlying task state reflects where it executed. Testing both worlds checks whether a representation supports required execution while respecting effects already present, rather than simply encouraging repetition.

Within each world, five *renderings* retain the same history and complete scope facts (Table 1). All retain the binding and directory before history. The local pointer copies the focal call/environment identifiers beside its receipt. The local full join also copies the environment's purpose and current connection from the directory. For example, it combines "call $c$ belongs to environment B" with "B is the rehearsal instance." This deterministic lookup makes existing scope explicit beside the receipt, adding no observation, completion label or execution instruction. The *end full join* places the same copy in a user-side metadata message immediately before the resumption request. Removing verified copies and wrappers exactly recovers each world's distributed input.

Table 1: Five presentation controls, each applied in both execution worlds. All retain the complete mapping before history, native receipt content and available tools. The two ordinary controls are separate. Copies add no new facts.

| Condition | JSON wrapper | Copied scope information | Copy location |
|---|---|---|---|
| Distributed | No | None | None |
| Neutral wrapper | Yes | None | None |
| Local pointer | Yes | Call/environment IDs | Beside receipt |
| Local full join | Yes | IDs, purpose, current connection | Beside receipt |
| End full join | No | IDs, purpose, current connection | Before resumption |

Two ordinary controls provide a task-capability baseline without cross-environment metadata. They retain the same task specification, with histories ending immediately before or after the focal operation in the task environment. The *pre-execution* control tests whether the agent can produce the required effect. The *post-execution* control tests continuation with an effect already present, including unnecessary repetition. We distinguish repeated calls from extra native effects: a repeated call may be rejected or leave state unchanged.

### 3.4 Collection and endpoints

We evaluate the 13 Azure deployments named in Table 2. Each task/deployment pair receives ten scope conditions and two ordinary controls: $60 \times 13 \times (2 \times 5 + 2) = 9{,}360$ assignments, with

one trajectory per task–deployment–condition combination. Models choose whether to call tools or respond directly. Episodes allow 12 assistant turns, 24 tool calls and 120 seconds per request; the 1,800-second deadline is checked before each model turn. Only verified technical failures are retried, retaining all attempts and selecting the first technically valid result. Behavioral errors are not rerun. Appendix D details inference settings, compatibility adaptations and protocol failures.

We measure focal effects, official whole-task success and completion claims separately. Native state determines effects independently of model text; focal success need not imply whole-task success. Let $S_i \in \{0, 1\}$ indicate whether episode $i$ ends with the focal effect present in the task environment. Checks accept source-supported semantic equivalences, such as official amount tolerances (Appendix B). The target is the tool's immediate effect, such as recording a return request, not later shipping or settlement.

For a terminal response labeled Yes or No, $A_i = 1$ denotes an assertion that the focal effect is present in the task environment; otherwise $A_i = 0$. A false terminal effect claim is

$$F_i = A_i(1 - S_i). \tag{1}$$

An unfinished operation reported as pending has $S_i = 0$ but $F_i = 0$: avoiding a false completion claim is distinct from completing the operation. Requests for confirmation or verification, and transfer-to-human announcements alone, do not set $A_i = 1$. False-claim rates divide affirmative terminal claims contradicted by final focal state by all assigned cells in the condition. Unclear and no-terminal cases remain outside the binary definition of $A_i$ and are not counted as correct dialogue. They enter the rate denominator, but not the false-claim numerator. Unclear-assignment bounds are reported separately.

All 9,034 nonempty terminal responses receive Yes/No/Unclear claim labels; the other 326 assignments have no terminal text. Human validation compares two independent human label sets with GPT-5.6-sol on 904 stratified responses spanning all 60 tasks. Exact agreement is 94.80–98.78% before adjudication. Sol labels the remaining 8,130 responses under a boundary-clarified rubric. Appendix C reports class-wise comparisons and human adjudication.

Comparisons pair the same tasks within each deployment, including tasks whose ordinary controls fail. Pointwise 95% intervals use a task-cluster bootstrap: each resampled task retains its paired conditions and the fixed deployment panel. Coarser $\tau$ operation-block and annotation sensitivities appear in Appendix B.

# 4 Results

## 4.1 Success elsewhere induces false completion claims

Successful rehearsal history with distributed scope sharply increases false completion claims on both public sources. On $\tau$, the rate rises from 1.9% (9/468) in the ordinary pre-execution control to 79.7% (373/468) under distributed scope: a paired increase of 77.8 percentage points (95% interval [72.4, 82.7]). On AppWorld, it rises from 7.7% (24/312) to 69.9% (218/312), an increase of 62.2 points [50.3, 72.8]. These are claims contradicted by the operation's final state in the task environment, not failures inferred from a broader task score.

Every one of the 13 deployments shows this increase on both sources (Table 2). GPT-5.5 has 20/36 distributed false claims on $\tau$ and 12/24 on AppWorld; GPT-5.6-sol has 34/36 and 16/24, respectively, versus zero in their ordinary pre-execution controls. DeepSeek, Kimi, Grok and gpt-oss deployments reproduce the direction under the same scope transformation.

These errors remain at the end of native continuations in which agents can query current state and execute the required operation. A false completion response can terminate the episode without eliciting corrective tool feedback. In a separately audited roaming continuation, GPT-5.4 even reads the current disabled state but reports completion without an enabling call. This example is outside the 60-task evaluation counts (Appendix C).

## 4.2 Local scope binding improves execution and reporting

Copying the existing execution-scope relation beside the receipt reduces false claims to 18.6% (87/468) on $\tau$ and 24.7% (77/312) on AppWorld. The paired reductions are 61.1 percentage points

Table 2: False completion claims (%, ↓): 13 deployments, 60 tasks. Three primary conditions: Pre (pre-execution), Dist (distributed scope), Join (local full join). Dist/Join follow rehearsal success; the task-environment effect starts absent. Overall pools $\tau$/AppWorld 36:24. Aggregate denominators per condition: 468/312/780 for $\tau$/AppWorld/Overall, including no-terminal cases.

| | $\tau$ (36 tasks) | | | AppWorld (24 tasks) | | | Overall (60 tasks) | | |
|---|---|---|---|---|---|---|---|---|---|
| Deployment | Pre | Dist | Join | Pre | Dist | Join | Pre | Dist | Join |
| GPT-5.4-mini | 5.6 | 88.9 | 69.4 | 29.2 | 79.2 | 62.5 | 15.0 | 85.0 | 66.7 |
| GPT-5.4 | 0.0 | 94.4 | 2.8 | 4.2 | 79.2 | 0.0 | 1.7 | 88.3 | 1.7 |
| GPT-5.5 | 0.0 | 55.6 | 0.0 | 0.0 | 50.0 | 0.0 | 0.0 | 53.3 | 0.0 |
| GPT-5.6-luna | 0.0 | 91.7 | 0.0 | 8.3 | 75.0 | 4.2 | 3.3 | 85.0 | 1.7 |
| GPT-5.6-terra | 0.0 | 80.6 | 0.0 | 0.0 | 70.8 | 4.2 | 0.0 | 76.7 | 1.7 |
| GPT-5.6-sol | 0.0 | 94.4 | 0.0 | 0.0 | 66.7 | 4.2 | 0.0 | 83.3 | 1.7 |
| DeepSeek-V4-Flash | 0.0 | 91.7 | 22.2 | 0.0 | 58.3 | 37.5 | 0.0 | 78.3 | 28.3 |
| DeepSeek-V4-Pro | 2.8 | 97.2 | 44.4 | 0.0 | 75.0 | 33.3 | 1.7 | 88.3 | 40.0 |
| Kimi-K2.5 | 8.3 | 86.1 | 11.1 | 4.2 | 58.3 | 25.0 | 6.7 | 75.0 | 16.7 |
| Kimi-K2.6 | 0.0 | 41.7 | 0.0 | 0.0 | 37.5 | 8.3 | 0.0 | 40.0 | 3.3 |
| grok-4 | 2.8 | 55.6 | 0.0 | 20.8 | 91.7 | 8.3 | 10.0 | 70.0 | 3.3 |
| grok-4-1-fast-non-reasoning | 2.8 | 83.3 | 69.4 | 8.3 | 79.2 | 62.5 | 5.0 | 81.7 | 66.7 |
| gpt-oss-120b | 2.8 | 75.0 | 22.2 | 25.0 | 87.5 | 70.8 | 11.7 | 80.0 | 41.7 |
| **Aggregate** | **1.9** | **79.7** | **18.6** | **7.7** | **69.9** | **24.7** | **4.2** | **75.8** | **21.0** |

(95% interval [56.4, 65.8]) and 45.2 points [35.6, 54.5], respectively. Both scope conditions contain the same facts; the intervention changes their attachment to the receipt. The reduction persists across task subsets and when compatibility-adapted assignments are excluded (Appendix B).

Native execution improves alongside reporting (Table 3): focal-effect completion rises from 15.0% to 75.2% on $\tau$ and from 25.3% to 71.2% on AppWorld. Ordinary pre-execution controls reach 88.7% and 87.2%, respectively. The local join therefore closes much, but not all, of the execution gap.

The benchmarks' official evaluators also register gains in whole-task success. On $\tau$, success rises from 15.0% under distributed scope to 74.1% with the local full join, compared with 88.0% in the pre-execution control. On AppWorld, the corresponding rates are 22.1%, 64.7% and 66.0%. These state-based outcomes do not depend on terminal-claim annotation. They show that the benefit extends beyond avoiding an incorrect assertion or satisfying the designated focal operation to completing the public task in the checkpoint-continuation setting. Appendix B provides the full deployment-level counts.

A neutral JSON wrapper leaves false-claim rates near the distributed condition: 77.8% on $\tau$ and 67.3% on AppWorld (Table 3). The local conditions use the same wrapper, but the large reduction appears when existing scope bindings are attached to the receipt.

Table 3: Reporting and execution after rehearsal success, with the task-environment effect initially absent. All entries are percentages: false completion claims (↓), native focal-effect completion (↑) and official whole-task success (↑). Each condition has 468 $\tau$ and 312 AppWorld assignments. All five renderings retain the complete mapping; Pre-execution is the ordinary control. Whole-task success uses the original benchmark evaluator.

| | $\tau$ ($n = 468$) | | | AppWorld ($n = 312$) | | |
|---|---|---|---|---|---|---|
| Condition | False claim ↓ | Focal effect ↑ | Whole task ↑ | False claim ↓ | Focal effect ↑ | Whole task ↑ |
| Pre-execution | 1.9 | 88.7 | 88.0 | 7.7 | 87.2 | 66.0 |
| Distributed | 79.7 | 15.0 | 15.0 | 69.9 | 25.3 | 22.1 |
| Neutral wrapper | 77.8 | 15.2 | 15.0 | 67.3 | 29.2 | 26.3 |
| Local pointer | 28.4 | 66.5 | 65.4 | 27.6 | 69.6 | 61.2 |
| Local full join | 18.6 | 75.2 | 74.1 | 24.7 | 71.2 | 64.7 |
| End full join | 54.7 | 38.9 | 38.9 | 55.8 | 40.1 | 36.2 |

Paired outcomes distinguish removed from newly introduced false claims (Table 4). For the same task/deployment pairs, local joining removes 288 false completion claims and introduces two on $\tau$; it removes 149 and introduces eight on AppWorld. Thus, reductions within matched pairs substantially outnumber newly introduced false claims. A removed false claim need not mean successful execution: an agent can instead acknowledge non-completion. Table 3 measures execution separately.

Table 4: Paired distributed/local-join false claims. Removed: false only under distributed; introduced: false only under local join. Both/neither refer to this endpoint, not task success. Reduction is the net decrease in percentage points, with task-cluster 95% intervals. Unclear and no-terminal cases are not counted as false claims.

| Source | Pairs | Removed | Introduced | Both | Neither | Reduction (pp) [95% CI] |
|---|---|---|---|---|---|---|
| $\tau$ | 468 | 288 | 2 | 85 | 93 | 61.1 [56.4, 65.8] |
| AppWorld | 312 | 149 | 8 | 69 | 86 | 45.2 [35.6, 54.5] |

The short local pointer alone lowers false claims to 28.4% on $\tau$ and 27.6% on AppWorld. A nearby call-to-environment identifier thus helps even while the full environment directory remains elsewhere in the history. Expanding the lookup further reduces aggregate errors, although it is not uniformly better than the pointer for every deployment (Appendix B).

Repeating the full relation at the end also helps in aggregate, but leaves false-claim rates of 54.7% on $\tau$ and 55.8% on AppWorld, substantially above the local full join. Channel, distance and formatting differ between these renderings, so the contrast identifies a representation effect rather than position alone.

Responses to local binding differ across deployments: GPT-5.5 has zero observed local-join false claims on both sources, while GPT-5.4-mini retains 25/36 on $\tau$ and 15/24 on AppWorld.

**Sensitivity to claim annotation.** Because the validation sample is Yes-dominated, high overall human–judge agreement alone is insufficient to assess the effect of labeling disagreements. Replacing adjudicated validation-sample labels with the original labels from the first human annotation set changes each of the four principal contrasts (scope-induced increases and local-join reductions on the two sources) by at most 0.32 percentage points. Both effect directions also persist under either assignment of every Unclear response (Appendix B). The main comparisons are therefore stable to the observed annotation disagreements and ambiguous labels.

### 4.3 Local binding preserves already-completed effects

The already-executed world and post-execution controls test whether the intervention merely encourages repetition. In the already-executed world, all 780 local-full-join episodes end with the required focal effect satisfied. For 377 of these episodes, native semantics also support measuring effects beyond the required amount; none has a positive final surplus. Two AppWorld continuations repeat the exact call but are rejected without state change. Full-condition results distinguish unnecessary calls, final focal-state violations and extra effects (Appendix B).

## 5 Controlled Diagnostics

Eight synthetic tasks test two explanations for premature action: interpreting mode records as intentions, and mistaking a tool whose name suggests verification for an unrestricted query. Each task requires a live effect before a dependent tool can complete the workflow. A successful preview leaves that effect absent, so immediately calling the dependent tool violates its precondition. Mode evidence remains linked to the focal call but appears earlier in the history, separately from its result. Making this remote evidence explicitly describe actual rather than intended execution leaves GPT-5.4 with 8/8 first-action errors. A separate control changes only the baseline tool's verification-suggesting name to a neutral completion name, preserving its description and precondition; 7/8 errors remain. Neither intention ambiguity nor the tool name alone accounts for these errors.

Copying actual-mode evidence beside its result reduces GPT-5.4's first-action errors from 8/8 to 3/8; repeating it at the end leaves 6/8. Because the copies also differ in message role and format, this contrast concerns representation rather than position alone. As in the public tasks, local binding of existing execution evidence improves continuation, here by reducing premature tool actions. Appendix E reports all conditions and models.

GPT-5.5 makes no observed first-action errors in the synthetic panel, yet produces distributed-scope false terminal claims on 20/36 $\tau$ tasks and 12/24 AppWorld tasks. In the diagnostic environment, all 47 trajectories with initial precondition violations recover. The environment rejects the premature tool call and returns a deterministic precondition error, after which the agent performs the missing operation and completes the workflow. In the public protocol, a completion claim can instead end the episode without automatic precondition feedback. Action selection, recovery after feedback and the truth of terminal completion claims are therefore distinct evaluation targets.

## 6 Discussion

**Relation availability does not ensure correct use.** Every scope rendering is relationally complete: truthful receipts and the supplied call-to-environment mapping establish where historical effects occurred. Yet agents can leave the current operation undone and report it complete. Local binding of the same scope information improves both execution and reporting. The central challenge is therefore to apply available relations when deciding what to do and what can truthfully be claimed complete.

**Keep execution scope attached to its receipt.** Copy the existing call-to-environment relation beside the receipt it qualifies, without adding a new state observation or a completion label. A short pointer reduces false completion claims; the full join further improves aggregate execution and reporting on both sources. These gains support local binding as a useful representation intervention, not a correctness guarantee. Native effects and reporting must remain separate evaluation targets: an episode without a false completion claim may still leave the requested effect absent.

**Fresh observations must inform completion claims.** Making a query available, issuing it and using its answer are separate steps. The audited roaming case shows a completion claim contradicting a fresh read, with no intervening enabling action (Appendix C). Evaluations should check completion claims against observed state and any intervening state changes, not merely count verification calls. A safeguard that checks only proposed writes can miss this omission-and-claim failure. A next step is to test completion-time checks that compare declarations with current-environment evidence before allowing termination.

## 7 Conclusion

Genuine success in another environment can leave LLM agents reporting completion without performing the requested operation in the target environment. On 60 public-derived tasks from two native systems, successful rehearsal history with distributed scope information raises false completion claim rates relative to matched pre-execution controls. The increase occurs in all 13 tested deployments on both sources, despite complete scope information and native tools. Copying existing scope bindings beside receipts substantially reduces false completion claims and improves completion of the required operation on both sources. Reliable continuation requires applying execution-scope relations to actions and completion claims, not merely retaining truthful history and complete scope information.

## 8 Limitations

We study selected benchmark tasks and deployments, not production failure rates or a representative sample of LLMs. Continuations start from fixed checkpoints without further user dialogue, in isolated environments with truthful scope metadata.

Shared $\tau$ customers and templates limit task independence; sensitivity checks also resample domain/focal-API blocks (Appendix B). One run per task/deployment/condition leaves repeated-

run variability unmeasured. Human validation is limited by the Yes-dominated sample and few No or Unclear labels (Appendix C). Some AppWorld focal-state criteria were revised after collection against native task semantics, retaining the original scoring view (Appendix B).

Local and end-of-history renderings differ in length, message channel, format and distance, so their contrast does not isolate position. Native interfaces differ across sources, so cross-source rate differences cannot be attributed to tasks alone. Four AppWorld tasks for gpt-oss-120b use a schema adapter; excluding these task/deployment pairs preserves the principal effect directions (Appendix B). Surplus-effect conclusions cover only tasks where native semantics support their measurement.

## Ethics Statement

All task operations were executed in isolated benchmark environments, without affecting real-user accounts. No external participants were recruited, and no real-user personal data were collected.

## AI Use Statement

Under author direction, AI assisted with code development, literature search, exploratory research design and analysis, preliminary drafting, and figure preparation. GPT-5.6-sol annotated terminal claims (human validation and adjudication: Appendix C). Authors take final responsibility for the research and submitted content.

## Reproducibility Statement

Appendices A, D and F document benchmark versions and adaptations, collection, recovery and artifact traceability. The anonymized supplementary package contains code, per-episode metrics and checksum-verified aggregate results for offline reanalysis and figure/table rebuilding. Raw responses and restricted benchmark content are excluded.

## REFERENCES


Ander Alvarez, Santhiya Rajan, Alessandro Genuardi, Oliver Wirjadi, Samuel Mugel, and Román Orús. ProvenanceGuard: Source-aware factuality verification for MCP-based LLM agents. *arXiv preprint arXiv:2606.18037v3*, 2026. doi: 10.48550/arXiv.2606.18037. URL https://arxiv.org/abs/2606.18037v3.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. doi: 10.48550/arXiv.2506.07982. URL https://arxiv.org/abs/2506.07982.

EnvTrustBench Contributors. EnvTrustBench: Evaluating LLM agents' evidence-grounding robustness at scale, 2026. URL https://envtrustbench.github.io/EnvTrustBench/. Public project description. Accessed September 15, 2026; manuscript and individual author metadata withheld.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/59f6421e64707225fdf5b28840679a07-Abstract-Conference.html.

LangChain. Persistence. LangGraph Documentation, 2026. URL https://docs.langchain.com/oss/python/langgraph/persistence. Documentation snapshot accessed September 19, 2026.

Junchi Liao. Auditing provenance sensitivity in LLM agent action selection. *arXiv preprint arXiv:2607.20827*, 2026. doi: 10.48550/arXiv.2607.20827. URL https://arxiv.org/abs/2607.20827.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Stripe. Sandboxes. Stripe Documentation, 2026. URL https://docs.stripe.com/sandboxes. Documentation snapshot accessed September 19, 2026.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. doi: 10.48550/arXiv.2406.12045. URL https://arxiv.org/abs/2406.12045.

Yu Zhuang, Kefei Chen, Yitong Duan, Shuxin Zheng, Jian Li, and Xu-Yao Zhang. AgentRewind: Recoverable execution for long-horizon LLM agents. *arXiv preprint arXiv:2608.14380*, 2026. doi: 10.48550/arXiv.2608.14380. URL https://arxiv.org/abs/2608.14380.

# A Public Sources, Selection and Checkpoint Adaptations

We use public tasks from Sierra's $\tau$ benchmark infrastructure (Barres et al., 2025) and AppWorld (Trivedi et al., 2024), with fixed source revisions and AppWorld data version 0.2.0.[1] The frozen task manifest records each source task, original instruction, reference actions, focal arguments and adapted checkpoint messages (Appendix F).

**Task selection.** The 36 $\tau$ tasks comprise 10 airline, 14 retail and 12 telecom instances across ten focal APIs. Twelve are the lowest-identifier eligible cases in 12 declared task/operation strata, excluding a three-task development pilot. Twenty come from a structurally screened source-task inventory. The remaining four are the lexicographically first unused eligible case in each of four retail operation strata. Selection uses native executability and focal-effect qualification, not target-model success or failure. All trajectories in the 60-task panel were collected anew.

AppWorld contributes 24 structurally qualified tasks, each from a distinct task-generator family: 16 from the benchmark's training split and eight from its development split. Focal applications are file system (four), phone (two), notes (one), Spotify (eleven) and Venmo (six). The reference-history prefixes supplied at resumption contain six to 213 calls. All 60 task identities and their selection provenance are listed in the task manifest.

**Native qualification and checkpoints.** For each task, the reference trajectory passes the native evaluator, omitting the focal write fails it, and restoring that write recovers success. Qualification verifies instance isolation, authentic receipts and the focal effect's absence or presence in native state. It also checks state-query availability and the effects of repeated actions. Agents need not reproduce the reference path. Because $\tau$ tasks can share customers, policies and troubleshooting templates, Appendix B also reports sensitivity to resampling coarser domain/focal-API blocks.

Before continuation, the $\tau$ checkpoint supplies reference-supported customer choices and approval. Airline adaptations specify itinerary, cabin, budget or payment choices while retaining additional requests and policy restrictions. Retail adaptations specify orders, variants, payment methods or destination addresses while retaining other requested operations. Telecom supplies recorded user-side device actions and approved roaming or data top-up choices, without inventing a later device test or successful speed-test result. Approval authorizes the operation; it does not certify execution. Within each task, the checkpoint request and approval are identical across conditions, including ordinary controls.

AppWorld preserves the original user instruction verbatim, without an additional approval exchange. When models query API documentation, the interface adapter registers the discovered APIs' native schemas for subsequent calls. Continued actions are model-selected: the runner does not append reference actions, invent user replies or repair credentials automatically. In contrast, the $\tau$ interface exposes full domain menus directly: 14 airline, 16 retail and 13 telecom tools.

**Rendering integrity.** Each task yields 12 inputs: five scope renderings in each of two worlds, plus two ordinary controls, giving 720 inputs across 60 tasks before deployment assignment. Every scope rendering retains the complete call-to-environment binding and the directory of environment purposes and current connection. Automated checks remove only protocol-specified copies and wrappers and exactly recover the corresponding distributed input. The intervention changes the presentation and local attachment of existing scope information, preserving genuine receipts and complete relations.

# B Full Results and Robustness Checks

Tables 5–7 report all rehearsal-world renderings and the ordinary pre-execution control. Each deployment–condition combination has 36 assignments on $\tau$ and 24 on AppWorld. All assigned tasks are included, including those whose pre-execution controls fail. Model names identify the tested Azure deployments; architecture and training are not controlled experimental factors.

[1] Public upstream revisions: $\tau$: `a2c0247` and AppWorld: `a072b7a`. The $\tau$ snapshot announces $\tau^3$ while retaining the `tau2` package and repository name.

Table 5: Complete false-claim counts by source: each $\tau$ cell is out of 36 assignments; each AppWorld cell is out of 24. Pre: pre-execution; Dist: distributed; Wrap: neutral wrapper; Ptr: local pointer; Join: local full join; End: end full join. The gpt-oss-120b AppWorld panel includes four compatibility-adapted tasks.

| | $\tau$: 36 assignments per cell | | | | | | AppWorld: 24 assignments per cell | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Deployment | Pre | Dist | Wrap | Ptr | Join | End | Pre | Dist | Wrap | Ptr | Join | End |
| GPT-5.4-mini | 2 | 32 | 31 | 35 | 25 | 32 | 7 | 19 | 20 | 18 | 15 | 21 |
| GPT-5.4 | 0 | 34 | 33 | 2 | 1 | 35 | 1 | 19 | 18 | 2 | 0 | 14 |
| GPT-5.5 | 0 | 20 | 20 | 0 | 0 | 6 | 0 | 12 | 10 | 0 | 0 | 6 |
| GPT-5.6-luna | 0 | 33 | 31 | 0 | 0 | 25 | 2 | 18 | 18 | 0 | 1 | 16 |
| GPT-5.6-terra | 0 | 29 | 26 | 0 | 0 | 12 | 0 | 17 | 17 | 0 | 1 | 9 |
| GPT-5.6-sol | 0 | 34 | 30 | 0 | 0 | 17 | 0 | 16 | 11 | 0 | 1 | 12 |
| DeepSeek-V4-Flash | 0 | 33 | 34 | 17 | 8 | 31 | 0 | 14 | 15 | 9 | 9 | 15 |
| DeepSeek-V4-Pro | 1 | 35 | 33 | 25 | 16 | 34 | 0 | 18 | 19 | 10 | 8 | 19 |
| Kimi-K2.5 | 3 | 31 | 28 | 9 | 4 | 8 | 1 | 14 | 14 | 10 | 6 | 10 |
| Kimi-K2.6 | 0 | 15 | 19 | 2 | 0 | 2 | 0 | 9 | 12 | 1 | 2 | 5 |
| grok-4 | 1 | 20 | 23 | 0 | 0 | 1 | 5 | 22 | 17 | 3 | 2 | 9 |
| grok-4-1-fast-non-reasoning | 1 | 30 | 31 | 26 | 25 | 31 | 2 | 19 | 18 | 17 | 15 | 19 |
| gpt-oss-120b | 1 | 27 | 25 | 17 | 8 | 22 | 6 | 21 | 21 | 16 | 17 | 19 |

Table 6: Native outcomes on $\tau$, each count out of 36 assignments. Separate panels report focal-effect completion and official whole-task success; completing the focal operation need not complete the whole task. Columns match Table 5.

| | (a) Native focal-effect completion | | | | | | (b) Official whole-task success | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Deployment | Pre | Dist | Wrap | Ptr | Join | End | Pre | Dist | Wrap | Ptr | Join | End |
| GPT-5.4-mini | 29 | 3 | 5 | 1 | 7 | 3 | 27 | 3 | 5 | 1 | 7 | 3 |
| GPT-5.4 | 34 | 1 | 1 | 28 | 32 | 1 | 33 | 1 | 1 | 27 | 30 | 1 |
| GPT-5.5 | 35 | 15 | 14 | 36 | 36 | 29 | 35 | 15 | 14 | 36 | 36 | 29 |
| GPT-5.6-luna | 36 | 3 | 5 | 36 | 35 | 10 | 36 | 3 | 5 | 33 | 34 | 10 |
| GPT-5.6-terra | 36 | 5 | 8 | 36 | 36 | 23 | 36 | 5 | 8 | 36 | 35 | 23 |
| GPT-5.6-sol | 36 | 1 | 5 | 36 | 36 | 18 | 36 | 1 | 5 | 36 | 36 | 18 |
| DeepSeek-V4-Flash | 35 | 2 | 1 | 17 | 27 | 4 | 35 | 2 | 1 | 17 | 27 | 4 |
| DeepSeek-V4-Pro | 35 | 0 | 1 | 10 | 20 | 2 | 35 | 0 | 1 | 9 | 20 | 2 |
| Kimi-K2.5 | 22 | 4 | 6 | 26 | 31 | 27 | 22 | 4 | 5 | 26 | 30 | 27 |
| Kimi-K2.6 | 24 | 21 | 14 | 34 | 36 | 32 | 24 | 21 | 14 | 34 | 36 | 32 |
| grok-4 | 31 | 15 | 11 | 36 | 36 | 33 | 31 | 15 | 11 | 36 | 36 | 33 |
| grok-4-1-fast-non-reasoning | 33 | 0 | 0 | 3 | 2 | 0 | 33 | 0 | 0 | 3 | 2 | 0 |
| gpt-oss-120b | 29 | 0 | 0 | 12 | 18 | 0 | 29 | 0 | 0 | 12 | 18 | 0 |

Table 7: Native outcomes on AppWorld, each count out of 24 assignments. The panels distinguish focal-effect completion from official whole-task success. Columns match Table 5.

| | (a) Native focal-effect completion | | | | | | (b) Official whole-task success | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Deployment | Pre | Dist | Wrap | Ptr | Join | End | Pre | Dist | Wrap | Ptr | Join | End |
| GPT-5.4-mini | 14 | 2 | 3 | 2 | 7 | 1 | 10 | 2 | 3 | 2 | 6 | 1 |
| GPT-5.4 | 22 | 5 | 5 | 22 | 24 | 8 | 21 | 4 | 4 | 21 | 23 | 8 |
| GPT-5.5 | 24 | 12 | 14 | 24 | 24 | 18 | 24 | 12 | 14 | 24 | 24 | 18 |
| GPT-5.6-luna | 21 | 6 | 6 | 24 | 22 | 7 | 19 | 6 | 6 | 22 | 22 | 7 |
| GPT-5.6-terra | 24 | 7 | 7 | 24 | 23 | 15 | 24 | 7 | 7 | 24 | 23 | 14 |
| GPT-5.6-sol | 24 | 8 | 13 | 24 | 23 | 12 | 24 | 8 | 12 | 24 | 23 | 12 |
| DeepSeek-V4-Flash | 24 | 9 | 9 | 14 | 14 | 7 | 12 | 5 | 8 | 8 | 7 | 6 |
| DeepSeek-V4-Pro | 22 | 5 | 4 | 13 | 16 | 5 | 10 | 3 | 2 | 5 | 12 | 4 |
| Kimi-K2.5 | 21 | 4 | 5 | 13 | 15 | 11 | 14 | 3 | 5 | 12 | 13 | 8 |
| Kimi-K2.6 | 21 | 13 | 12 | 23 | 19 | 18 | 11 | 12 | 11 | 19 | 17 | 14 |
| grok-4 | 18 | 2 | 7 | 21 | 22 | 15 | 17 | 2 | 7 | 20 | 22 | 15 |
| grok-4-1-fast-non-reasoning | 21 | 3 | 3 | 5 | 7 | 4 | 15 | 3 | 3 | 5 | 6 | 4 |
| gpt-oss-120b | 16 | 3 | 3 | 8 | 6 | 4 | 5 | 2 | 0 | 5 | 4 | 2 |

**Task pairing and uncertainty.** Pointwise 95% intervals use 10,000 task-cluster bootstrap resamples with seed 20260925. Each resampled task retains all deployments and paired conditions. The intervals quantify between-task variation within the fixed deployment panel. They do not estimate production prevalence, variation across a model population or repeated-run variability within a cell. Table 4 reports both false claims removed and false claims introduced by the local full join.

The coarser $\tau$ sensitivity groups tasks by domain/focal API into ten blocks. Its intervals are [70.1, 85.8] percentage points for the increase from pre-execution to distributed scope and [58.1, 64.3] for the local-join reduction. Both effect directions persist under this coarser resampling. Table 8 separates the 12 tasks selected from declared task/operation strata from the other 24 eligible selections (Appendix A). A separate AppWorld comparison excludes the four compatibility-adapted task/deployment pairs. The selection subsets use the same collection and are not held-out splits.

Table 8: Subset sensitivity: false-claim rates with explicit per-condition denominators. Pre, Dist and Join denote pre-execution, distributed scope and local full join. The 48 adapted AppWorld assignments span four task/deployment pairs and all 12 conditions; exclusion removes four pairs per displayed condition.

| Subset | Pairs | Pre (%) | Dist (%) | Join (%) |
|---|---|---|---|---|
| *Task-selection subsets* | | | | |
| $\tau$: declared strata (12 tasks) | 156 | 2.6 | 80.8 | 17.3 |
| $\tau$: other eligible selections (24 tasks) | 312 | 1.6 | 79.2 | 19.2 |
| *Interface-adapter sensitivity* | | | | |
| AppWorld: all | 312 | 7.7 | 69.9 | 24.7 |
| AppWorld: adapter excluded | 308 | 7.1 | 69.5 | 23.7 |

Allowing either label assignment for each Unclear response bounds the increase from pre-execution to distributed scope at 77.8–78.0 percentage points on $\tau$ and 61.9–62.2 on AppWorld. The corresponding local-join reductions range from 60.9–61.3 and 44.9–45.2 points. These are assignment bounds, not confidence intervals. Restoring original human labels in the validation sample changes any of the four principal contrasts by at most 0.32 points. Using the judge's original sample labels leaves all four point estimates unchanged.

**Already-executed controls.** The initially completed group comprises 4,680 assignments across the five task-world renderings and ordinary post-execution control. Table 9 separates exact-call repetition, unsatisfied final focal state and positive final surplus. Three final focal violations comprise

two $\tau$ excess data-top-up episodes under neutral wrapping and one AppWorld end-join playback-state violation. Six positive-surplus episodes comprise those two top-up cases and four AppWorld payment-request cases. These totals overlap; they are not nine distinct failures.

Table 9: Already-executed controls, shown as affected episodes / applicable assignments. Unsatisfied: final focal effect unsatisfied; exact repeat: focal call repeated with identical arguments; surplus: positive final excess where measurable. Surplus has a smaller denominator; unsupported cases are unknown, not zero. Post denotes ordinary post-execution; other condition labels follow Table 5.

| Source | Condition | Unsatisfied | Exact repeat | Surplus |
|---|---|---|---|---|
| $\tau$ | Post | 0/468 | 1/468 | 0/169 |
| $\tau$ | Dist | 0/468 | 0/468 | 0/169 |
| $\tau$ | Wrap | 2/468 | 3/468 | 2/169 |
| $\tau$ | Ptr | 0/468 | 0/468 | 0/169 |
| $\tau$ | Join | 0/468 | 0/468 | 0/169 |
| $\tau$ | End | 0/468 | 0/468 | 0/169 |
| AppWorld | Post | 0/312 | 0/312 | 1/208 |
| AppWorld | Dist | 0/312 | 1/312 | 1/208 |
| AppWorld | Wrap | 0/312 | 3/312 | 1/208 |
| AppWorld | Ptr | 0/312 | 2/312 | 0/208 |
| AppWorld | Join | 0/312 | 2/312 | 0/208 |
| AppWorld | End | 1/312 | 2/312 | 1/208 |

The local full join retains all 780 focal effects and has no positive final surplus among 377 supported assignments. Two AppWorld exact-call repetitions are rejected without state change. Conversely, semantically duplicate effects can arise from calls with different argument values, so surplus can occur without an exact-call repeat.

**Source-supported semantic state.** A post-collection audit inspected all 60 tasks against source instructions, evaluator semantics and native transitions. All 36 $\tau$ predicates remain unchanged. Twelve AppWorld focal-state rules were revised to recognize source-supported equivalent outcomes. The rules allow text normalization and official amount tolerances, disregard incidental timestamps, and recognize valid playlist states. For file compression, they check the archived payload rather than requiring a matching generated binary placeholder. Identity, ownership and required effects remain checked. These revisions were made after collection and frozen before uniform reanalysis of the saved states.

Uniform application reclassifies 42 final focal states from unsatisfied to satisfied: 32 payment-share and ten file-compression episodes, comprising 27 rehearsal and 15 completed-world assignments. The surplus audit identifies three additional duplicate-effect episodes, already included in the six reported above. Original predicates and their scores are preserved separately. All official whole-task outcomes remain unchanged.

Completing the focal operation need not complete the whole task: AppWorld pre-execution yields 272/312 focal effects but 206/312 official successes. Avoiding a false claim is also distinct from completing the operation: GPT-5.6-luna's local join has no $\tau$ false claims but leaves one of 36 focal effects unsatisfied. An unsatisfied final predicate alone does not prove that the operation never occurred; an effect may overshoot its authorized amount or be undone later.

## C Claim Annotation and Validation

The claim rubric asks whether the final response asserts that the designated focal effect is present in the current task environment. Yes denotes an explicit or clearly entailed assertion of that effect; No denotes no such assertion, including explicit statements of non-completion. Unclear denotes ambiguity in the target, execution scope or assertion attribution. The target is the tool's immediate effect: claiming submission of a return request qualifies as Yes without claiming completed shipping or settlement. Statements about other operations, rehearsal-only success, verification requests and

handoff announcements alone do not qualify. A false completion claim is identified when a Yes label is combined with an unsatisfied final focal state.

**Sample and independent labels.** The annotation pool contains 9,034 nonempty terminal responses. A frozen sample of 904 is drawn proportionally across source, deployment, world and arm strata using largest-remainder quotas and uniform within-stratum sampling (seed 20260924). Its 592 $\tau$ and 312 AppWorld responses cover all 60 tasks and 13 deployments. Two human annotators (H1 and H2) and GPT-5.6-sol label the same 904 responses independently under the original rubric, without access to one another's labels or adjudication decisions. Each receives the focal operation and arguments, original request, complete model-visible context and final response. Deployment and condition labels, prior annotations and native audit outcomes are masked. Ordinary receipts and scope context remain visible and can reveal the rendering.

Table 10: Original independent labels from two human annotators on the same 904 responses, compared with GPT-5.6-sol before adjudication. Rows are human labels; columns are GPT-5.6-sol labels. Agreement is not accuracy against a known gold standard.

| Annotator | Human \ GPT-5.6-sol | Yes | No | Unclear | Total |
|---|---|---|---|---|---|
| H1 | Yes | 880 | 7 | 0 | 887 |
| | No | 0 | 13 | 0 | 13 |
| | Unclear | 1 | 3 | 0 | 4 |
| H2 | Yes | 846 | 5 | 0 | 851 |
| | No | 10 | 11 | 0 | 21 |
| | Unclear | 25 | 7 | 0 | 32 |
| Each | Total | 881 | 23 | 0 | 904 |

Table 10 reports exact three-class agreement with the judge of 893/904 (98.78%) for H1 and 857/904 (94.80%) for H2. Human–human agreement is 861/904 (95.24%). An always-Yes labeler agrees with H1 on 887/904 (98.12%) and H2 on 851/904 (94.14%), illustrating label imbalance. The judge matches all 13 H1 No labels and 11 of 21 H2 No labels; H2 uses Unclear more often (32 versus four for H1).

Human adjudication of the 11 H1–judge disagreements accepts eight corrections to No; three boundary cases yield two No and one Unclear under the clarified rules below. This produces 880 Yes, 23 No and one Unclear in the sample used for the final analysis. Author review of all 47 H2–judge disagreements yields 35 Yes and 12 No, with no further changes to the analysis labels. The review distinguishes focal-operation completion from unfinished non-focal goals and subsequent verification. Both human label sets and the judge's original labels remain separate; all agreement figures use labels before adjudication.

**Clarifications and remaining labels.** Diagnostic metadata and quoted status text are not outward answers. A diagnostic wrapper without an outward answer is No. A bare verification request is No, whereas an explicit completion assertion followed by a test request remains Yes. A broad outcome that only ambiguously implies the focal operation is Unclear. GPT-5.6-sol labels the remaining 8,130 responses with these clarifications. Initial agreement evaluates the original rubric; it is not a held-out test of the revised prompt. Appendix B reports sensitivity to original sample labels and Unclear assignments.

Final labels are 8,730 Yes, 286 No and 18 Unclear. The other 326 assignments have no terminal text and remain in full-schedule denominators without fabricated semantic labels. No agent trajectory is rerun for annotation.

Eight sample quotations require Markdown/escape normalization; all eight labels match H1 (format-strict agreement: 885/896). Including them yields the all-904 H1 comparison above. The remainder's 258 quotation-to-source alignment cases are resolved offline through fragment matching and individual assistant inspection, without label changes or resampled judgments. Original judge quo-

tations are preserved with separate source-text mappings. These checks establish quotation provenance, not semantic label accuracy.

**Supplementary continuation cases.** The introduction's roaming example is a separately collected case, excluded from the 60-task evaluation counts. Its genuine rehearsal receipt reports success; a current-instance lookup reports roaming disabled, and no enabling call follows. GPT-5.4 reports completion and attributes the contrary lookup to synchronization delay. Saved inputs, tool traces and responses support the audit.

A separate airline-upgrade audit examines ten transfer-to-human summaries from one task with verified unsatisfied final focal states. Eight falsely assert the upgrade, including four without a false terminal response. These cases are also excluded from the 60-task counts: the main endpoint evaluates final responses, excluding intermediate and handoff assertions.

## D Collection, Integrity and Technical Recovery

The evaluation comprises 5,616 $\tau$ and 3,744 AppWorld assignments: 720 per deployment and 9,360 episodes overall. Each task–deployment–condition combination contributes one episode. Returned model identifiers, request configurations and attempt histories are archived.

Automatic tool selection uses native source-specific interfaces (Appendix A), with parallel tool calls disabled within an episode. Requests do not explicitly set temperature, reasoning effort or an output-token limit, leaving these to deployment defaults. Episode limits are 12 assistant turns and 24 native calls, a 120-second request timeout and a 1,800-second deadline checked before each model turn. Collection uses three to ten concurrent episodes; prompts, per-episode inference settings, scoring and task selection are unchanged. Request-start events are durably saved before dispatch. Native tools run in isolated benchmark environments; only model inference uses external services. No reference continuation or simulated user reply follows the experimental checkpoint.

**Recovery and coverage.** All 9,360 assignments are accounted for: 9,359 outcomes pass execution-integrity verification and one non-retryable Kimi-K2.6 model-protocol failure is retained, leaving no unresolved technical gap. Verification does not imply task success. The audit preserves 9,499 attempts and 26,067 provider-request starts, not additional independent episodes. Interface diagnostics are excluded from experimental results. Verified interruptions, transport failures and native infrastructure failures allow at most three total attempts, including the initial attempt. The first technically valid result is retained; incorrect answers, refusals, empty responses, exhausted budgets and model-generated invalid arguments do not justify retries. Where possible, execution is reconstructed offline from unchanged saved responses without further model calls.

**Compatibility adapter.** Forty-eight gpt-oss-120b/AppWorld assignments (four tasks, all 12 conditions) use an adapter removing `default: null` from enumerated parameter schemas. Only the model-visible schema annotation changes; natural-language messages, tool names, allowed values, native implementations, inference budgets and scoring remain unchanged. Original incompatible requests and errors are retained as interface failures, not model-behavior failures. Exclusion sensitivity removes all four task/deployment pairs (Table 8). The adapter does not modify model-generated arguments.

**Native verification.** Offline replay checks saved responses and calls against fixed environment versions, state journals and official outcomes. AppWorld's official evaluator scores saved databases in an isolated process; their hashes must remain unchanged after scoring. Actions, state and individual test outcomes must agree; diagnostic-wording variations are recorded separately. Original attempts, mappings from episodes to selected attempts, replay reports and archive hashes are preserved. Scoring revisions (Appendix B) and annotation adjudications (Appendix C) change derived measurements only, not native trajectories.

## E Controlled Diagnostic Tests

Eight synthetic development tasks, selected by a fixed catalog rule across eight platform families, yield 336 episodes: three GPT deployments, two states, seven conditions, one episode per combination. The completion tool requires a live effect; premature calls return `precondition_failed` and permit further action. Episodes allow four model turns and a 90-second request timeout; failed requests are not retried.

The original trace places truthful mode evidence before the tool history, linked to the focal call. Conditions vary actual-mode wording, result-local copying, final repetition, read-only query availability or completion-tool name, preserving goals and state-transition rules (Table 11).

Table 11: First-action precondition errors after preview success (live effect absent), each out of eight tasks. Public-task terminal claims are evaluated separately.

| Condition | GPT-5.4 | GPT-5.5 | GPT-5.6-luna |
|---|---|---|---|
| Original truthful trace | 8 | 0 | 3 |
| Explicit actual remote receipt | 8 | 0 | 2 |
| Original + local mode copy | 1 | 0 | 0 |
| Actual remote + local mode copy | 3 | 0 | 0 |
| Actual remote + final repetition | 6 | 0 | 0 |
| Original + independent query | 8 | 0 | 0 |
| Original + neutral completion name | 7 | 0 | 1 |

All 47 first-action errors (41 GPT-5.4, six GPT-5.6-luna) recover after precondition feedback. GPT-5.5 has no first-action errors in this panel. The already-committed world yields 167 correct first actions across 168 episodes; one request fails before answering. No model queries first in the query condition; GPT-5.4 makes three later queries. Overall, 329/336 episodes complete and seven API/network failures remain unknown. Offline auditing checks all 554 model requests and state transitions.

The actual-remote condition explicitly names and describes records and fields as post-execution evidence; GPT-5.4 still errs on 8/8 tasks. Renaming the completion tool while preserving its description and precondition leaves 7/8 errors. Neither intention ambiguity nor tool naming alone accounts for these errors. Result-local copying reduces errors to 3/8, versus 6/8 for final repetition, a comparison of carrier and position together. A final claim can end a public episode without automatic state-check feedback: recovery and terminal-claim correctness remain distinct endpoints.

## F Artifacts and Reproducibility

The archive separates frozen inputs, original and technical-recovery attempts, native audits, claim annotations and derived tables. Each task–condition input package identifies the public task, original request, checkpoint, native tools, scope mapping and input checksum. Offline rendering checks cover all 720 base inputs (60 tasks $\times$ 12 conditions); evaluating them across 13 deployments yields 9,360 episode assignments. Each assignment links to its retained attempt, saved response, native outcome or failure status, and claim annotation or no-terminal-text status.

Public-task result tables are generated from a fixed, checksum-verified aggregate export. The overall columns in Table 2 weight each task equally within a deployment and condition, pooling 36 $\tau$ and 24 AppWorld tasks. Per-episode records preserve task pairing, labels, technical failure and recovery status, and supporting quotations. Both sets of 904 original human annotations, original judge outputs, adjudications and clarified rubric remain separate (Appendix C); adjudication does not overwrite independent labels. Separately collected qualitative cases are excluded from the 60-task evaluation counts.

Documentation in the anonymized supplementary package lists dependencies and commands to rebuild tables, figures and the PDF, with consistency checks. Rebuilding uses saved analysis without new model calls or re-executing native tasks. The package excludes raw responses, credentials

and restricted benchmark content; full native replay requires the original checkpoint and trajectory artifacts.