SOURCE-MSIS: TRACING SOURCE ERRORS IN INTERACTIVE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Accurately tracing root causes is key to helping interactive agents recover from failures in long-horizon tasks. Observations at failure may reveal only the cascading effects of earlier, unnoticed errors. Existing methods can flag suspicious steps in execution traces, but struggle to verify root causes and derive precise, nonredundant recovery plans. We propose the Source-indexed Minimal Sufficient Intervention Sequence (SOURCE-MSIS) framework for tracing source errors and verifying recovery. The framework uses execution records from failed tasks to identify earlier errors that may block later steps and proposes targeted repairs for these errors. It then replays the task to check whether the repairs remove the corresponding blockages. If the task remains incomplete, feedback from the new execution guides the search for other errors and additional repairs. Once the task is completed, an independent verifier removes one or more repairs and replays the remaining repairs in their original order under the same conditions. A plan passes verification only if the full plan completes the task and every reduced plan fails to do so. Across three interactive benchmarks with publicly available base models, SOURCE-MSIS achieves an overall recovery success rate of 65.3%, compared with 5.7% for the strongest baseline, and reaches 100% on ALFWorld. These results show that tracing source errors and verifying recovery through replay provide concrete targets for repair, helping agents complete previously failed tasks.



Figure 1: SOURCE-MSIS identifies planning errors made by a Gemma3-12B agent in Habitat and replays targeted repairs to complete the tasks. Red, amber, and green borders mark failure, replay with repairs, and task completion, respectively.

1 INTRODUCTION

Consider an embodied agent tasked with transporting two objects. It leaves one in a cabinet, takes only the other to the destination, and then exhausts its action budget. The final observation shows that the task is incomplete, but does not explain why the second object was not transported: the agent may have overlooked this goal, or may not have been given actions for opening the cabinet or picking up the object. Different causes require different corrections. Failure diagnosis therefore needs to trace the trajectory to identify which task prerequisite was not met and test whether correcting it

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.

allows execution to continue. We study this localization of source errors and whether it can help agents recover from failed tasks.

Similar problems occur in tool use, web interaction, embodied systems, and multi-agent systems (Liu et al., 2024; Wang et al., 2024; Zhou et al., 2024; Mialon et al., 2024; Chang et al., 2025; Duan et al., 2025; Cemri et al., 2025; Zhu et al., 2025). Existing methods can locate suspicious steps in a trajectory and improve subsequent execution through retries, reflection, or replay (Yao et al., 2023; Shinn et al., 2023; Madaan et al., 2023; Xu et al., 2025; Zhang et al., 2025; 2026; Nagalapatti et al., 2025; Lin et al., 2026; Ma et al., 2026). However, suspicious steps still need to be connected to specific recovery actions. In the transport task above, restoring the pickup action may still leave the agent overlooking the second object; correcting one condition is not enough to complete the task. Diagnosis must therefore address multiple blockages in the same trajectory and, after recovery, examine which corrections contributed to it.

We propose SOURCE-MSIS, a framework that traces source errors to diagnose failures and guide recovery. It uses state changes, action feedback, and task dependencies in failed trajectories to propose candidate sources. It tests each candidate in the original task to see whether correcting it allows the agent to make progress. If a trajectory contains multiple errors, the method checks task progress after each correction to find blockages that become apparent later. After recovery, it removes one or more interventions from the repair sequence and reruns the task to check whether it can still be completed. The resulting diagnosis explains why the agent failed and provides the corrections needed for execution to continue.

We compare Source-MSIS with four existing methods for failure diagnosis and repair in Habitat, ScienceWorld, and ALFWorld, examining whether it helps agents recover more failed tasks with different base models. Figure 2 illustrates the idea through a case with two source errors.

Our main contributions are as follows:

- We propose a framework that traces source errors to diagnose failures and guide recovery, linking evidence in failed trajectories to executable repairs and their local effects;
- We design an ordered intervention search and an independent verification algorithm, using trajectories after repair to find subsequent source errors and deletion tests to identify redundant interventions in recovery sequences;
- Experiments in three interactive environments demonstrate the framework's recovery performance: Source-MSIS outperforms the comparison methods in ScienceWorld and ALFWorld across all three model configurations, while its performance in Habitat varies with the model configuration.

2 RELATED WORK

Failure localization and attribution. Failure localization studies where an execution trajectory begins to go wrong. AHA combines visual and language evidence to detect robotic manipulation failures; MAST categorizes failure modes in multi-agent systems; AgentDebug traces responsible modules through cascading errors (Duan et al., 2025; Cemri et al., 2025; Zhu et al., 2025). OpenRCA uses telemetry to identify multiple software root causes, while Who&When and AgenTracer locate responsible agents and critical steps (Xu et al., 2025; Zhang et al., 2025; 2026). Aegis and AgentRx study failure attribution using data from injected errors and evidence from trajectories, respectively (Kong et al., 2026; Barke et al., 2026). Source-MSIS links candidate sources to local repairs and tests whether those repairs remove the corresponding blockages during execution.

Intervention-based diagnosis and repair. Intervention-based methods test diagnoses by changing execution. IDI repairs anomalous root nodes; REFLECT and DoVer use controlled patches or targeted edits. CausalFlow and CAR link the steps responsible for failure to counterfactual repairs and replay (Nagalapatti et al., 2025; Lin et al., 2026; Ma et al., 2026; Bonagiri et al., 2026; Shah, 2026). AgentTether and HarnessFix also turn trajectory evidence into guidance with a defined scope or local patches (Zhao et al., 2026; Chen et al., 2026). Source-MSIS also tests diagnoses through execution. It orders repairs for multiple source errors by task dependency, then checks whether recovery remains possible after removing some repairs.

Recovery and reflection agents. ReAct alternates reasoning and action, while Reflexion stores feedback to improve subsequent attempts (Yao et al., 2023; Shinn et al., 2023). Self-Refine and CRITIC revise outputs using model or tool feedback, while SCoRe and AgentRefine directly train self-correction capabilities (Madaan et al., 2023; Gou et al., 2024; Kumar et al., 2025; Fu et al., 2025). PARTNR studies recovery in embodied multi-agent tasks (Chang et al., 2025). Source-MSIS also records which source error each repair targets. It independently tests deletions from successful sequences to identify repairs that are indispensable within those sequences.

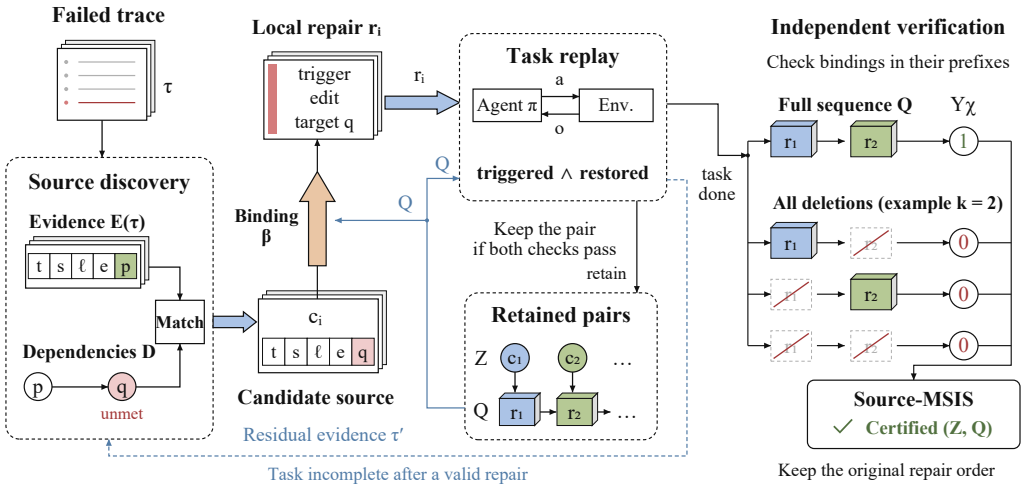

Figure 2: SOURCE-MSIS traces sources in a failed trajectory, binds repairs, and continues searching with replay feedback. An independent verifier checks whether the full repair sequence restores the task and whether shorter order-preserving subsequences can complete the same task.

3 SOURCE-MSIS: FROM SOURCE ERROR LOCALIZATION TO INTERACTIVE RECOVERY

Completing a long-horizon interactive task usually requires an interactive AI system to perform several interdependent steps. Whether later steps can be executed depends on the states established earlier, the task prerequisites already satisfied, and the actions currently available. When one of these conditions fails, subsequent steps that depend on it may go wrong or stall (Ross et al., 2011; Zhu et al., 2025). This section first describes how source errors and their interventions are represented, then defines the conditions for a minimal sufficient intervention sequence and explains how Source-MSIS searches for and tests such sequences.

3.1 SOURCE ERRORS: WHAT WE LOCATE IN A FAILED TRAJECTORY

At each step, the agent receives an observation o_t, and its policy π selects an action a_t based on the task and interaction history. The environment executes the action and returns a new observation. Here, the policy includes the model, prompt, and decision rules used to generate actions; observations include state summaries, task progress, and interface feedback visible to the method. An action and the observations before and after it form an event record, whose step index indicates execution order. We denote a trajectory that fails to complete the task by

$$\tau_f = (o_0, a_0, o_1, \ldots, a_{T-1}, o_T), \qquad \Omega(\tau_f) = 0, \tag{1}$$

where T is the number of steps at the end of execution, and Ω is the task evaluator. A value of 0 indicates that the task success conditions have not been met. State records, action results, and interface information that can be retrieved from the trajectory constitute diagnostic evidence.

When a task fails, the trajectory usually shows directly only the step at which the agent stopped. Consider the task of putting an apple in a refrigerator. The agent reaches the refrigerator but cannot place the apple, so the end of the trajectory records a failed placement action. Reviewing earlier execution reveals that, when the agent passed the apple's location, the action interface did not offer

the policy the action "take apple." The agent therefore reached the refrigerator without holding the apple. The failed placement records where the task stopped, whereas the earlier interface omission explains why the prerequisite of holding the apple was not met. Understanding this failure requires both the preceding trajectory and the task dependencies between steps.

We call a failure of an earlier condition that prevents subsequent task dependencies from being satisfied a *source error*, as illustrated above by the unavailable pickup action. A source error occurs at a specific time and state, lies at a particular layer of the agent or interaction system, and blocks subsequent task progress through a specific error event. Accordingly, we represent a source error instance as

$$c_i = (t_i, s_i, \ell_i, e_i, q_i), \tag{2}$$

where t_i is the step at which evidence of the error appears. We use this index to check whether the error precedes the affected steps. The public state record s_i describes the state of that event, since the same action may serve different task functions in different states. The system layer ℓ_i, such as policy behavior, state recording, or the action interface, specifies where subsequent checks should be made. The field e_i records the specific error that occurred, such as omitting an available action or recording an incorrect object state. The task milestone q_i identifies which part of the task the error prevents the agent from completing. Together, these five fields identify a source error instance. We record errors separately when they occur at different times or states, or block different milestones. In the previous example, "an error occurred at step 12" does not specify the state, system layer, or consequence. A complete description is: "at step 12, while the apple was available for pickup, the action interface did not expose the pickup action to the policy, preventing the subsequent holding milestone from being satisfied."

A failed case may contain multiple source error instances. Long-horizon tasks typically involve several connected prerequisites. For example, when transporting two objects, picking up and placing each object form separate task dependencies. Missing the pickup of the first object does not rule out errors in the interface, planning, or state records involving the second. Errors may also become apparent in sequence. An earlier error may prevent the agent from reaching a later subtask, so the next blocked dependency becomes visible only after that error is corrected (Zhang et al., 2025; Zhu et al., 2025). Even if two errors share the same mechanism, we record them separately when they occur at different times or states, or block different task milestones.

Source-MSIS compares task requirements with execution records, working backward from unmet milestones through their prerequisites to events that may have disrupted them (Reiter, 1987; Zhang et al., 2025; Zhu et al., 2025). We organize anomalies in the trajectory into a finite set of evidence records $\mathcal{E}(\tau_f)$. Each record (t, s, ℓ, e, p) contains the event location, public state, system layer, suspected error, and the prerequisite p it may affect. The record also links to the original log segment supporting it. Let $\mathcal{U}(\tau_f)$ be the set of unmet milestones. In the task dependency relation $\mathcal{D}$, a pair (p, q) means that completing milestone q requires condition p, possibly through intermediate steps. Matching evidence records to these dependencies gives the candidate set:

$$\mathcal{C}^{(0)} = \big\{(t, s, \ell, e, q) \mid q \in \mathcal{U}(\tau_f),\ \exists p \left[(t, s, \ell, e, p) \in \mathcal{E}(\tau_f) \wedge (p, q) \in \mathcal{D}\right]\big\}. \tag{3}$$

For example, an unavailable pickup action may prevent the agent from holding the apple, which is required for subsequent placement. This connection forms a source error hypothesis for testing. Equation (3) records both the anomaly's location and the affected milestone; subsequent interventions test whether the proposed connection holds. The algorithm's $\mathrm{Propose}$ function performs this construction, and new replay records update the candidates. Evidence extraction and dependency matching in each environment are described in Appendix A.1.

All subsequent tests take place within the same failed case. Each replay holds the environment, task, initial state, original policy, success criterion, and execution time limit constant. When testing an additional repair, the method retains the preceding repairs in their existing order. Deletion tests remove only the selected repairs while keeping the remaining conditions unchanged. We call these shared comparison conditions a *fixed failure case*.

3.2 Source-Indexed Interventions: How We Test a Candidate Source

Evidence makes candidate c_i worth testing but does not establish whether it blocked the task. We bind an intervention to c_i and test its milestone q_i after the preceding repairs. Let χ denote the fixed

failure case and $\mathcal{R}$ its permitted local interventions. Write $Q_{<i} = (r_1, \ldots, r_{i-1})$ for the retained prefix, with $Q_{<1} = ()$. The local test is

$$\begin{aligned} r_i &= \beta(c_i; Q_{<i}) \in \mathcal{R}, \\ \tau_i^r &= \mathrm{Run}(\chi, Q_{<i} \parallel r_i), \\ \mathrm{ValidBind}_{\chi, Q_{<i}}(c_i, r_i) &:= \mathrm{Triggered}_{c_i}(\tau_i^r) \wedge \mathrm{Restored}_{q_i}(\tau_i^r). \end{aligned} \tag{4}$$

The binding function β constructs r_i for the error described by c_i, given the repairs already retained. It specifies when to intervene, what may change, and which observable condition establishes that q_i has been restored. Replay executes $Q_{<i}$ before applying r_i at its specified state; the same policy then continues the task. The symbol $:=$ denotes a definition. $\mathrm{Triggered}$ checks that r_i was executed at that state, and $\mathrm{Restored}$ checks the milestone immediately after the repair. Both checks must pass. For example, a pickup repair must leave the agent holding the apple. A later repair may depend on an earlier repair to reach its trigger state; it need not pass this test when executed alone.

3.3 Minimal Sufficient Intervention Sequences

Even after intervention r_i for candidate source c_i restores milestone q_i, the task may stop at a later step. The same case may contain other source errors, some of which become apparent only after an earlier intervention takes effect. We therefore arrange candidates and interventions that pass the local tests in their actual execution order. When a set of interventions eventually restores the task, we also check for redundancy: the full sequence should restore the task, while removing any one or more interventions should cause it to fail again. A sequence satisfying these two conditions is called a Minimal Sufficient Intervention Sequence (MSIS). Source-MSIS searches for such sequences while retaining the binding between each intervention and its candidate source.

Let $Z = (c_1, \ldots, c_k)$ denote candidate sources in execution order, with $k \geq 1$. Here, k counts the declared repairs, not individual environment actions or search rounds. The binding function constructs the intervention program sequentially:

$$Q = \beta(Z) = (r_1, \ldots, r_k), \qquad r_i = \beta(c_i; Q_{<i}) \in \mathcal{R}. \tag{5}$$

For a program P that can be replayed completely in fixed failure case χ, we write

$$Y_\chi(P) = \Omega(\mathrm{Run}(\chi, P)) \in \{0, 1\} \tag{6}$$

for the task outcome, where 1 denotes task completion and 0 indicates that the task remains incomplete after program execution. Removing at least one intervention from Q while preserving the relative order of the rest yields a program denoted by $Q' \sqsubset Q$. This includes the empty program obtained by removing all interventions. Source-MSIS accepts an ordered pair (Z, Q) of source errors and an intervention sequence if it satisfies three conditions: valid local bindings, sufficiency of the full sequence, and minimality of the sequence:

$$\begin{aligned} &\mathrm{ValidBind}_{\chi, Q_{<i}}(c_i, r_i), && \forall i \in \{1, \ldots, k\}, \\ &Y_\chi(Q) = 1, && \\ &Y_\chi(Q') = 0, && \forall Q' \sqsubset Q. \end{aligned} \tag{7}$$

The first condition tests each binding after its prefix in the original sequence Q. The second establishes sufficiency, and the third establishes minimality. A deletion test does not require the remaining repairs to pass their original binding checks: removing a prerequisite repair may prevent a later repair from triggering. It tests whether the remaining program completes the task under the same case conditions. Here, "minimal" means that removing any one or more declared repairs prevents recovery, not that Q is globally shortest. For $Q = (r_1, r_2)$, the full sequence must succeed while $()$, (r_1), and (r_2) all fail. If (r_1) succeeds, Q is not minimal. All 2^k retained subsets, including the empty and full sequences, require outcome checks; Appendix A.6 specifies their accounting.

3.4 The Source-MSIS Procedure

The conditions above test a constructed sequence; finding one requires trying repairs. Algorithm 1 replays each proposed intervention after the repairs already retained and checks its binding in that context. Failed local tests discard the candidate. Otherwise, the new trajectory guides further search until the task completes or the search budget B_s is exhausted. After recovery, independent verification checks the bindings and deletion outcomes within budget B_v. Exhausted search returns UNRESOLVED; an incomplete verification returns BLOCKED, not a certificate.

Algorithm 1: Source-MSIS construction and independent verification

Input: case χ, failed rollout τ_f, dependencies $\mathcal{D}$, binding β, budgets B_s, B_v
Output: CERTIFIED, NON-MINIMAL, BLOCKED, or UNRESOLVED

$Z, Q \leftarrow (); \tau^{\text{res}} \leftarrow \tau_f$ *// source proposal and residual search*
$\mathcal{C} \leftarrow \text{Propose}(\tau^{\text{res}}, \mathcal{D})$ *// match evidence to dependencies*
while B_s remains and $\mathcal{C} \neq \varnothing$ **do**
  $c \leftarrow \text{Select}(\mathcal{C} \mid \tau^{\text{res}}, Q); r \leftarrow \beta(c; Q)$
  $(y, \tau') \leftarrow \text{Replay}(\chi, Q \parallel r)$ *// same case and agent*
  if the replay is not evaluable **then return** BLOCKED
  if $\neg\, \text{ValidBind}_{\chi,Q}(c, r)$ **then** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$; **continue**
  $Z \leftarrow Z \parallel c; Q \leftarrow Q \parallel r$
  if $y = 1$ **then return** $\text{Verify}(\chi, Z, Q, B_v)$
  $\tau^{\text{res}} \leftarrow \tau'; \mathcal{C} \leftarrow \text{Update}(\mathcal{C} \setminus \{c\}, \tau^{\text{res}}, \mathcal{D})$ *// use residual evidence*
end while
return UNRESOLVED
function VERIFY(χ, Z, Q, B_v) *// independent certificate check*
  $(y_Q, \{y_{Q'}\}) \leftarrow \text{ReplayAll}(\chi, Q, \{Q' : Q' \sqsubset Q\}, B_v)$
  if any replay is not evaluable **then return** BLOCKED
  $b \leftarrow \bigwedge_i \text{ValidBind}_{\chi,Q_{<i}}(c_i, r_i)$ *// check original prefixes*
  if $\neg b$ or $y_Q = 0$ **then return** UNRESOLVED
  if $\exists Q' \sqsubset Q : y_{Q'} = 1$ **then return** NON-MINIMAL
  return CERTIFIED(Z, Q) *// sufficient and subset-minimal*
end function

4 EXPERIMENTS AND ANALYSIS

Source-MSIS locates source errors and tests repairs by replaying the task. This section examines whether this process helps agents complete tasks that previously failed. We first describe the experimental tasks, base models, and comparison methods, using recovery success rate to compare the complete procedures. We then analyze the role of each component through ablations and examine where incomplete tasks stop progressing.

4.1 EXPERIMENTAL SETUP AND EVALUATION METRIC

We compare Source-MSIS with Who&When, AgenTracer, AgentDebug, CausalFlow, and DoVer. Who&When and AgenTracer locate responsible agents or steps; AgentDebug produces diagnoses and corrective feedback (Zhang et al., 2025; 2026; Zhu et al., 2025). CausalFlow tests interventions on trajectory dependencies, while DoVer tests failure hypotheses through targeted edits and replay (Bonagiri et al., 2026; Ma et al., 2026). The methods process the same failed cases and execute recovery plans in the original environments. Their adaptations to the environment interfaces are described in Appendix B.

Figure 3(a) shows the three experimental environments. Habitat contains 80 interactive 3D household rearrangement tasks covering object rearrangement, setting the table, tidying clutter, and restocking food. Methods read public task states and executable PDDL actions, and completion is determined by Habitat's rearrangement goals. ScienceWorld contains 150 long-horizon scientific tasks involving successive object manipulations and state changes. ALFWorld contains 110 household manipulation tasks that require agents to satisfy relationships among object states, containers, and action prerequisites (Szot et al., 2021; Wang et al., 2022; Shridhar et al., 2021). The three environments differ in trajectory length, action interfaces, and task dependencies. The original trajectories of all cases are judged unsuccessful by the respective official environment evaluators.

Each failed case is processed by all six methods under three model configurations: Qwen2.5-3B, Qwen2.5-7B, and Gemma3-12B. During recovery, the model participates in diagnosis, repair planning, or continued decision-making after intervention, with its specific role determined by the method's procedure. Actions in each plan are executed by the environment, and task outcomes are determined by the official evaluator. The original failure records serve as common inputs, and results for the three models are reported separately. Candidate construction and model roles in each environment are described in Appendix A.1.

The primary metric is Recovery Success Rate (RSR), which measures how often recovery completes the original task, as illustrated in Figure 3(b). For each method and model configuration in each environment, let N be the number of original failed cases evaluated and N_{succ} the number that pass the official task evaluator after repair execution. Then

$$\text{RSR} = \frac{N_{\text{succ}}}{N} \times 100\%. \tag{8}$$

We report successful case counts and RSR separately for each environment and model configuration. Cases without an executable plan or with an incomplete task after execution remain in the denominator. Task completion depends on the environment state; the method's textual judgments do not contribute to the score. Results from plan output through task evaluation are provided in Appendix B. We also examine whether the repairs satisfy the binding and minimality conditions. Appendix E reports the verification outcomes for individual cases and the replay costs.

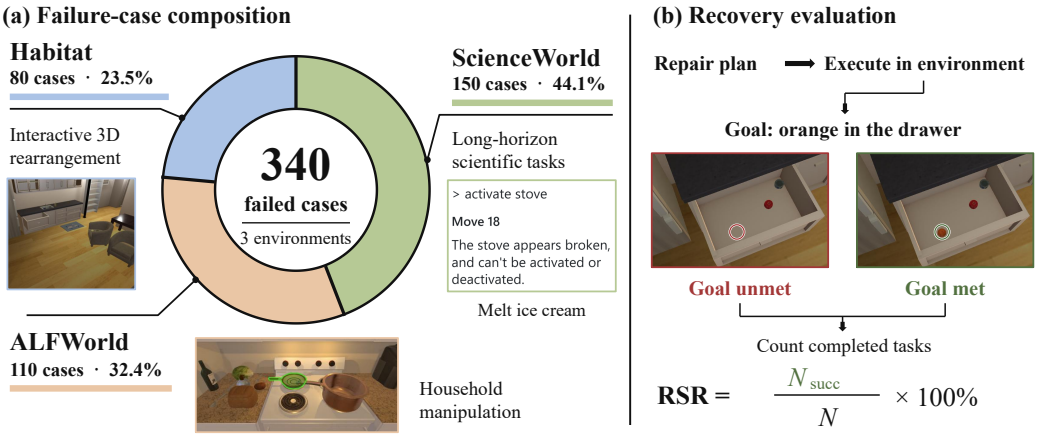

Figure 3: Composition of the 340 failed cases and calculation of recovery success rate (RSR). A case counts as recovered when it passes the environment's task evaluator.

4.2 Overall Recovery Results Across Models

Table 1 reports recovery results for six methods in three environments under three model configurations. Source-MSIS achieves the highest recovery success rate in every configuration of ScienceWorld and ALFWorld; its advantage in Habitat is mainly observed with the 7B and 12B configurations. We discuss these differences below.

Table 1: Recovery success rates of six methods on Habitat, ScienceWorld, and ALFWorld; each cell reports the number of successful cases and RSR.

Method	3B			7B			12B		
	Habitat	ScienceWorld	ALFWorld	Habitat	ScienceWorld	ALFWorld	Habitat	ScienceWorld	ALFWorld
Who&When	1/80 (1.3%)	0/150 (0.0%)	0/110 (0.0%)	4/80 (5.0%)	1/150 (0.7%)	1/110 (0.9%)	33/80 (41.3%)	0/150 (0.0%)	5/110 (4.5%)
AgenTracer	3/80 (3.8%)	0/150 (0.0%)	0/110 (0.0%)	6/80 (7.5%)	0/150 (0.0%)	1/110 (0.9%)	3/80 (3.8%)	0/150 (0.0%)	2/110 (1.8%)
AgentDebug	7/80 (8.8%)	0/150 (0.0%)	0/110 (0.0%)	2/80 (2.5%)	1/150 (0.7%)	1/110 (0.9%)	43/80 (53.8%)	0/150 (0.0%)	4/110 (3.6%)
CausalFlow	0/80 (0.0%)	0/150 (0.0%)	0/110 (0.0%)	4/80 (5.0%)	0/150 (0.0%)	0/110 (0.0%)	0/80 (0.0%)	0/150 (0.0%)	1/110 (0.9%)
DoVer	**10/80 (12.5%)**	0/150 (0.0%)	0/110 (0.0%)	6/80 (7.5%)	1/150 (0.7%)	2/110 (1.8%)	21/80 (26.3%)	0/150 (0.0%)	4/110 (3.6%)
SOURCE-MSIS	5/80 (6.3%)	**77/150 (51.3%)**	**110/110 (100.0%)**	**43/80 (53.8%)**	**78/150 (52.0%)**	**110/110 (100.0%)**	**54/80 (67.5%)**	**79/150 (52.7%)**	**110/110 (100.0%)**

Source-MSIS shows its clearest advantages in ScienceWorld and ALFWorld. In ScienceWorld, it completes 77/150, 78/150, and 79/150 tasks with the three models, respectively; under the same conditions, the comparison methods complete at most 1/150. In ALFWorld, Source-MSIS completes

all 110 tasks with each model, whereas the best comparison result is 5/110. This shows that, in these two environments, Source-MSIS is better able to turn localization results from failed trajectories into interventions that complete the task.

Switching to 7B or 12B models does not eliminate the recovery gap between the comparison methods and Source-MSIS in ScienceWorld and ALFWorld. Source error localization and intervention testing remain effective across all three model configurations.

Habitat shows a different trend. DoVer leads with the 3B model, completing 10/80 tasks compared with 5/80 for Source-MSIS. With the 7B and 12B models, Source-MSIS completes 43/80 and 54/80 tasks, while the best comparison methods complete 6/80 and 43/80. DoVer completes 44/1020 cases overall, including 37 in Habitat. Its addition leaves AgentDebug as the strongest baseline overall. Execution stages and paired comparisons are provided in Appendix B.

4.3 Ablation Experiments

We next remove components to examine which steps affect task completion. The components in Table 2 combine repairs for multiple sources, order them by their dependencies, and use replay feedback to find further errors.

Table 2: Source-MSIS component analysis, with differences in percentage points.

Component	Full method	Removed	Difference
ALFWorld: 7B			
Multi-source composition	110/110	5/110	**+95.5**
Dependency ordering	110/110	23/110	**+79.1**
MiniGrid, ScienceWorld (two-stage)			
Residual feedback	27/27	0/27	**+100.0**

In ALFWorld with 7B, the full method completes all 110 tasks, whereas retaining repairs for only one source completes just 5. Removing dependency ordering leaves only 23 of these tasks complete. These results show that recovery depends on combining repairs and satisfying the prerequisites for subsequent actions. With the same solver, removing historical source information from repair construction leaves ALFWorld completion unchanged under all three models (Appendix C.1).

Ordinary ALFWorld cases do not reveal the separate role of residual feedback: both the full method and the variant without it complete all 110 tasks. We therefore construct additional two-stage cases in which the second error becomes observable only after the first repair. Source-MSIS completes all 27 cases, whereas removing residual feedback leaves every case incomplete. The first repair lets execution continue and reveals another error, so the method needs the new feedback to find the next repair. Component settings, individual outcomes, and paired analyses are provided in Appendix C.

4.4 Recovery Across Repeated Runs

To test whether recovery is repeatable, we run 30 tasks from each environment independently five times with the three models and the original five methods. Each run starts from the same initial conditions, using different generation seeds at temperature 0.2. DoVer is evaluated in the main comparison only.

In Figure 4, Source-MSIS achieves an RSR of 63.7–66.7% per run, compared with at most 9.3% for any comparison method. It succeeds in all five runs for 151 combinations, versus at most 6 for a comparison method, so its advantage persists across repeated runs. Its task completion outcomes remain unchanged in ALFWorld and ScienceWorld; all 49 combinations with both successful and failed runs occur in Habitat. Detailed settings and results are provided in Appendix D.



Figure 4: Recovery success rates across five independent runs, each covering the same 90 tasks with three models. The broken y-axis shows 0–10% and 60–70%.

4.5 Why Tasks Remain Incomplete

Repeated runs show that recovery in Habitat still varies. Returning to individual records from the main experiment helps explain why repair plans fail. In Habitat, Source-MSIS completes 27 of the same tasks with both 7B and 12B but not with 3B. In one single-object transport case, the final 3B plan only approaches and picks up the object, leaving it in the gripper. Both 7B and 12B continue to the destination and place it. In another case requiring three grocery items to be moved, 3B places only the first item, whereas 7B and 12B move all three. These failures reflect an incomplete action sequence for one object and a plan that omits other task goals, respectively.

Such omissions occur beyond these examples. Among the 75 incomplete cases with 3B, 67 final plans omit placement for at least one object that has not reached its destination. Only two of the 37 incomplete cases with 7B have this omission, but 36 contain rejected actions in their final execution. Covering all task goals and executing the planned actions successfully therefore require separate attention during recovery. Incomplete ScienceWorld tasks also include runs without executable interventions; full Source-MSIS completes every ALFWorld task. Stage counts, case actions, goal states, and results across execution rounds are provided in Appendix B.

5 Conclusion

We study how to locate source errors in a failed interactive trajectory and test whether this localization helps an agent complete the task on another attempt. Source-MSIS proposes candidate sources from trajectory evidence, links them to local interventions, and lets the original agent continue in the same task environment. After recovery, the verifier deletes interventions from the sequence to check whether each change contributed to that recovery. Replaying the task tests both the proposed diagnosis and whether the corresponding repairs lead to recovery.

The applicability of Source-MSIS depends on whether failed cases can be replayed reliably and whether the intervention library can express the local changes required by the task. The method cannot complete its tests when the trajectory provides insufficient evidence or the required changes fall outside the intervention library. Future work needs to broaden the range of executable interventions so that the same source error tracing process can handle more open-ended interactive environments.

AI USE STATEMENT

We used generative AI tools to assist with experimental design and implementation, data preparation and analysis, language editing, and figure preparation. The authors take responsibility for the final content.

REPRODUCIBILITY STATEMENT

We provide supplementary data and interfaces to reproduce the recovery rates, paired comparisons, ablation results, statistics across repeated runs, and verification summaries reported in the paper. Experimental settings and evaluation procedures are described in the main text and appendices.

REFERENCES

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. Agentrx: Diagnosing ai agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. Causalflow: Causal attribution and counterfactual repair for llm agent failures. *arXiv preprint arXiv:2605.25338*, 2026.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A. Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, et al. PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations*, 2025.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A vision-language model for detecting and reasoning over failures in robotic manipulation. In *International Conference on Learning Representations*, 2025.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. AgentRefine: Enhancing agent generalization through refinement tuning. In *International Conference on Learning Representations*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations*, 2026.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025.

Xiaofeng Lin, Yingxu Wang, Tung Sum Thomas Kwok, Daniel Guo, Sahil Arun Nale, Charles Fleming, and Guang Cheng. REFLECT: Intervention-supported error attribution for silent failures in LLM agent traces. *arXiv preprint arXiv:2606.09071*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

Lokesh Nagalapatti, Ashutosh Srivastava, Sunita Sarawagi, and Amit Sharma. Robust root cause diagnosis using in-distribution interventions. In *International Conference on Learning Representations*, 2025.

Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi: 10.1016/0004-3702(87)90062-2.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

Jaineet Shah. Causal agent replay: Counterfactual attribution for llm-agent failures. *arXiv preprint arXiv:2606.08275*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimír Vondruš, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems*, volume 34, pp. 251–266, 2021.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024.

Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. OpenRCA: Can large language models locate the root cause of software failures? In *International Conference on Learning Representations*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations*, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599, 2025.

Chenyu Zhao, Shenglin Zhang, Wenwei Gu, Yongqian Sun, Dan Pei, Chetan Bansal, Saravan Rajmohan, and Minghua Ma. AgentTether: Graph-guided diagnosis and runtime intervention for reliable LLM agent operation. *arXiv preprint arXiv:2607.06273*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

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

A ALGORITHMIC DETAILS OF SOURCE-MSIS

Algorithm 1 in the main text gives the order in which Source-MSIS processes a failure. This section explains how the experiments construct candidates, generate repairs, and execute replays from public records in each environment. It also distinguishes task evaluation of the full program from subsequent certificate checks.

A.1 FROM TRAJECTORY EVIDENCE TO CANDIDATE SOURCES

The candidate constructor receives the task description, failed trajectory, public state, and action preconditions. Each candidate records at least an identifier, its corresponding trajectory interval, an error description, and the trajectory evidence supporting that judgment. The interval specifies where to review execution, while the evidence field retains the state, action result, or unmet task condition observed there. Subsequent ranking, repair generation, and result recording all reference the same candidate identifier.

Candidate construction uses public execution records and task dependencies, with a different implementation in each environment. Habitat provides the experimental model with the task, failed trajectory, current predicates, and goal predicates, asking it to identify trajectory intervals associated with blocked prerequisites and return evidence. The model then ranks candidates and generates repair actions for those retained. Public action prerequisites are used to check whether navigation, pickup, and placement actions connect correctly.

ScienceWorld first extracts locations where logs show rejected actions, repeated actions, or a lack of state progress. It then links the objects involved to unmet dependencies in public task progress to form candidate sources. The model participates in repair binding and action planning for these candidates. This process separately records blockages already visible in the logs and repairs that need to be tested through execution.

ALFWorld parses target objects, required quantities, and target states from the task description. It observes objects by navigating to public locations and exploring containers. It then constructs repairs according to the dependencies between picking up objects, changing their states, and placing them. Exploration produces new execution records that link repairs to specific objects and locations. The public task and observations guide program construction. The model also records an initial diagnosis and makes further decisions when needed after a repair. Exploration records retain the actions taken, observations received, and execution outcomes.

Success in all three environments is determined by actual environment execution. The experimental base models participate in diagnosis or recovery, with the original failure records serving as common inputs; the stage at which model calls occur varies with the implementations above.

Table 3: Main objects in Source-MSIS runtime records.

Object	Stored information
Candidate source	Candidate identifier, trajectory start and end positions, error description, and corresponding trajectory evidence.
Source–repair binding	Candidate identifier, checking order, replay starting point, public action sequence, and binding rationale.
Replay result	Actions actually executed, whether the environment returned an evaluable result, task outcome, and residual trajectory.
Final result	Task outcome of the full intervention sequence and the outcomes of its proper subsequences required for certificate checks.

A.2 RANKING, BINDING, AND ENVIRONMENT ACTIONS

The ranker combines candidate evidence, trajectory position, and repairs already retained to select the next candidate source to test. The binding function then determines the replay starting point and generates a local intervention using the environment's public action interface. Actions are checked against the interface before execution. Action names must come from the public action set for the current case, and sequence length must not exceed the case limit. Only programs that pass these checks are sent to the environment.

Different environments require different native actions, but the objects processed by the method remain the same. Habitat constructs PDDL actions from public initial and goal predicates and

evaluates task completion using `pddl_success`. ScienceWorld generates environment commands from the state ledger and public action grammar. ALFWorld produces household manipulation commands from public observations and the action interface. Environment adapters only translate intervention programs into actions executable in the corresponding environment and return the task outcomes supplied by that environment.

Table 4: States, interventions, and task evaluation in the three environments.

Environment	Public state	Intervention language	Task evaluation
Habitat	Initial predicates, goal predicates, and action preconditions	Executable PDDL actions	`pddl_success`
ScienceWorld	Failed trajectory, state ledger, and action grammar	ScienceWorld environment commands	Environment task outcome
ALFWorld	Failed trajectory, public observations, and action interface	ALFWorld household manipulation commands	Environment task outcome

A.3 Environment Replay and Residual Search

Each replay begins at the recovery starting point specified for the current case. The method first executes the ordered repairs already retained, then the intervention added in the current round, after which the original agent continues the task. The environment returns the task outcome, execution trajectory, and unmet task conditions. If the task is complete, the current program proceeds to certificate checking. Otherwise, Source-MSIS reads residual failure evidence from the new trajectory and passes both untested and new candidates to the next search round.

This process preserves the actual execution order of multiple repairs. Suppose the first two rounds select c_1 and c_2, with interventions r_1 and r_2. The second round executes (r_1, r_2), rather than only the newly added r_2, and checks $\mathrm{ValidBind}_{\chi,(r_1)}(c_2, r_2)$. If evidence of the second error appears only after r_1 takes effect, the error enters the candidate set in the second round. Independent verification checks this binding in the same prefix context. The separate deletion replay of (r_2) tests task completion, not whether the original trigger remains reachable after removing r_1.

A.4 Task Outcomes and Certificate Checks

The verifier evaluates the full sequence Q and every proper order-preserving subsequence $Q' \sqsubset Q$ under the same case conditions. Table 5 gives the four outputs of Algorithm 1. The verifier first checks whether the required replays are evaluable, then checks the bindings, full-sequence recovery, and minimality in that order.

Table 5: How Source-MSIS assigns its four output states.

Output	Decision from the recorded checks
Certified	The source–repair bindings pass their checks, the full sequence completes the task, and every proper order-preserving subsequence fails to complete it.
Non-Minimal	The bindings pass and the full sequence completes the task, but at least one proper order-preserving subsequence also completes it.
Blocked	A required replay cannot produce an evaluable result, so search or verification cannot reach a decision.
Unresolved	Search exhausts its candidates or budget without recovery, or independent verification rejects a binding or finds that the full sequence does not complete the task.

The main experiment uses RSR to compare task completion. Classification records and verification costs for the main experiment are reported in Appendix E. The controlled experiments are reported separately in Appendix C.5.

A.5 Shared Experimental Limits

The main recovery experiment permits at most 720 environment steps, 64 environment replays, 64 model calls, 8192 generated tokens, two retries for structured output, and 16 task evaluations per method and case. Methods that repeatedly generate repairs can perform at most five recovery rounds. An environment adapter may stop early when the task is complete, no new executable plan

is available, or its own stopping condition is met. These limits apply to producing the main task outcome; the subsequent independent checks of saved proposals are accounted for separately.

A.6 Verification Cost

For k declared repairs, there are $2^k - 1$ proper subsequences, including the empty sequence. Evaluating these subsequences and the full sequence requires 2^k outcome replays. A repair can contain multiple environment actions; its action count and the number of search rounds do not determine k. Repair boundaries are fixed before deletion testing.

Reusing the full-sequence result reduces the additional outcome replays to $2^k - 1$. Reusing both the full and empty results reduces them to $2^k - 2$, provided their replay conditions match exactly. An original failed trajectory does not by itself establish the empty-program result under a different continuation context. The saved-proposal verification reruns both endpoints. ALFWorld also executes the full repair program once to inspect local state transitions, giving $2^k + 1$ physical executions when all checks are completed. This inspection is not an additional subsequence.

The 16-evaluation limit does not guarantee exhaustive verification for arbitrary k: a sequence of five repairs requires 32 outcome replays. Algorithm 1 therefore records search and verification budgets separately. If a required check cannot finish within the verification budget, the procedure returns BLOCKED. Such a result does not establish minimality.

B Comparison Methods and Results at Each Execution Stage

Recovery success rate records whether a run ultimately completes the task. To show how this outcome is reached, we first describe how the comparison methods operate in the experiments, then divide each run into four successive stages: plan output, action submission, evaluable replay, and task success.

B.1 How Comparison Methods Execute in the Environment

All six methods receive the same types of public inputs: the task description, failed trajectory, recovery starting point, observable state, and available action interface. Who&When identifies the main failure location and responsible component in one pass over the full trajectory. The same base model then generates recovery actions from that diagnosis. AgentDebug first labels errors at each step using five module categories: Memory, Reflection, Planning, Action, and System. It then selects the earliest critical error and generates corrective feedback. AgenTracer uses the experimental base model to analyze candidate error steps, propose corrective actions for each, and replay the candidates chronologically. CausalFlow builds a sequential state-dependency graph from the trajectory and proposes local interventions on the graph. It replays the task in the environment to select interventions that change the outcome. Source-MSIS proposes candidate sources with trajectory evidence, binds repairs to them, and, when needed, continues searching using the residual trajectory after replay.

Actions generated by all methods are checked against the same public action interface. The environment executor accepts only action sequences that pass these checks. Diagnoses and plans are first written to the runtime records; the environment evaluator then computes RSR from actual execution outcomes. Table 6 summarizes how method outputs are connected to environment actions.

DoVer uses its released prompts for trial analysis, failure hypotheses, and targeted intervention generation (Ma et al., 2026). A simulator trace without explicit replanning messages is treated as one trial. The intervention replaces an instruction at the diagnosed step, and the same base model generates continuations through the public action interface. Earlier actions remain fixed. Each intervention is tested with three continuations, subject to the common execution limits. As for the other methods, RSR records whether any execution completes the task. Separately, DoVer marks an intervention as validated when all three executions are evaluable and at least two succeed. This criterion holds for 41 cases, compared with 44 recovered cases under RSR. The three additional recoveries occur in Habitat, one with 3B and two with 12B.

Table 6: Outputs of the comparison methods and how they are executed in the environment.

Method	Intermediate output	Recovery action generation	Selection in the environment
Who&When	Main failure step, responsible component, and failure description, identified in one pass	The base model generates a recovery sequence from the diagnosis and public action interface	Execute in rounds and stop when the task is complete
AgenTracer	Candidate error steps, original actions, corrective actions, and their rationale	Each candidate directly specifies corrective actions within the public interface	Replay in trajectory order and select the earliest successful candidate
AgentDebug	Module errors at each step and a critical error with corrective feedback	The base model generates a recovery sequence from the critical error and public action interface	Execute in rounds and stop when the task is complete
CausalFlow	Sequential state-dependency graph and local interventions on the graph	Each intervention directly specifies an action sequence within the public interface	Replay candidates; among successful ones, prefer fewer actions and earlier positions
DoVer	Failure hypothesis and replacement instruction at a diagnosed step	The base model resumes from that step using the edited instruction and public action interface	Test each edit with three continuations and record task success and majority validation
Source-MSIS	Candidate sources with supporting evidence, source–repair bindings, and ordered intervention programs	Construct a complete recovery sequence from candidate sources and the public action interface	Update the residual trajectory after replay and begin verification once the task is complete

B.2 Stage Definitions

"Plan output" means that a run saves both diagnosis and plan records. "Submitted" means that the executor records at least one recovery attempt, including attempts rejected by action checks. "Evaluable" means that at least one replay returns an environment result usable for task evaluation. "Success" means that the state after replay passes the environment's task evaluator. The first three columns describe how method outputs reach the environment; the last is the numerator of RSR in the main text.

The three environments comprise 6120 case–method–model combinations: $80 \times 6 \times 3 = 1440$ for Habitat, 2700 for ScienceWorld, and 1980 for ALFWorld. Method outputs and termination records are saved for all combinations, and all 6120 runs have completed. Table 7 reports the case counts at each stage.

The overall rate in the abstract pools runs across environments and model configurations. It divides the total number of successes by the total number of runs for each method.

Table 7: Case counts at each execution stage in the three main experiments; each cell lists plan outputs, submitted cases, evaluable cases, and successes, in that order.

Model	Method	Habitat	ScienceWorld	ALFWorld
3B	Who&When	80/80/80/1	150/150/150/0	110/110/60/0
	AgenTracer	80/80/80/3	150/150/150/0	110/110/103/0
	AgentDebug	80/80/80/7	150/149/149/0	110/110/97/0
	CausalFlow	80/80/80/0	150/150/150/0	110/110/107/0
	DoVer	80/80/80/10	150/91/91/0	110/101/98/0
	Source-MSIS	80/80/80/5	150/130/130/77	110/110/110/110
7B	Who&When	80/80/80/4	150/150/150/1	110/110/105/1
	AgenTracer	80/80/80/6	150/147/147/0	110/110/104/1
	AgentDebug	80/80/80/2	150/149/149/1	110/109/107/1
	CausalFlow	80/80/80/4	150/149/149/0	110/110/105/0
	DoVer	80/61/61/6	150/112/112/1	110/90/86/2
	Source-MSIS	80/80/80/43	150/135/135/78	110/110/110/110
12B	Who&When	80/80/80/33	150/148/148/0	110/110/103/5
	AgenTracer	80/80/80/3	150/150/150/0	110/110/105/2
	AgentDebug	80/80/80/43	150/141/141/0	110/110/103/4
	CausalFlow	80/80/80/0	150/148/148/0	110/110/98/1
	DoVer	80/79/79/21	150/118/118/0	110/99/77/4
	Source-MSIS	80/80/80/54	150/119/119/79	110/110/110/110

In Habitat, 1420 of 1440 runs produce evaluable results. DoVer has 19 cases with 7B and one with 12B that stop before submitting an executable continuation. Success counts are low for all

methods with the 3B model; Source-MSIS completes 43 and 54 tasks with the 7B and 12B models, respectively.

In ScienceWorld, the five comparison methods save 150 outputs under each model configuration and produce evaluable replays for 91 to 150 cases, but complete at most one task. Source-MSIS produces 130, 135, and 119 evaluable programs, of which 77, 78, and 79 complete the task, respectively.

In ALFWorld, the comparison methods produce evaluable replays for 60 to 107 cases under each model, with final success counts ranging from 0 to 5. All 330 Source-MSIS runs produce evaluable results and complete the task.

B.3 Goal Omissions and Action Execution in Habitat

We compare the original Source-MSIS execution records for each case across the three model configurations. Of the same 80 tasks, 27 succeed with both 7B and 12B but fail with 3B. The main text uses two of these cases to illustrate an incomplete action sequence for one object and omitted goals for multiple objects.

The single-object transport case requires placing the object at its destination and leaving the gripper empty. After two execution rounds, the final 3B plan still only approaches and picks up the object. Both actions are accepted, but the object remains in the gripper. In its second round, 7B executes four steps: approach the object, pick it up, move to the destination, and place it. The 12B model executes the same four-step plan in its first round, and both configurations satisfy the final goals.

The grocery case requires moving three objects to their respective destinations. After three rounds, the final 3B plan still addresses only the first object. That object reaches its destination, but the other two goals remain unmet. Both 7B and 12B pick up, transport, and place all three objects in their first round. These examples correspond to the fourth single-object rearrangement case and the third grocery case in the original task records.

Table 8 summarizes execution across all 80 tasks. An omission is counted when an object remains away from its destination and the final plan contains no placement action for that object and destination. Rejected actions are read directly from the environment records. A failed case may contain both an omission and a rejected action, so these rows are not additive. The remaining rows count cases reaching a second or later execution round and those that ultimately succeed within this group.

Table 8: Goal coverage, action execution, and recovery across rounds for Source-MSIS in Habitat.

Execution record	**3B**	**7B**	**12B**
Incomplete tasks	75	37	26
Of these: placement omitted for an unmet goal	67	2	19
Of these: rejected action in the final execution	42	36	14
Cases reaching a second or later round	69	51	12
Of these: ultimately successful	3	14	6

Most incomplete 3B tasks omit actions needed for their goals. Plans from 7B usually include the corresponding placement actions, but execution can still fail. Incomplete 12B tasks also include both omissions and rejected actions. Together with the examples, these records show the need to cover every object's goal and order pickup, navigation, and placement so they can execute successfully.

B.4 Per-Case Paired Tests

Methods face the same tasks within each environment and model size, so statistical comparisons are paired by task. We use two-sided exact McNemar tests between Source-MSIS and each comparison method, applying Holm correction to the five comparisons within each environment and model size. Table 9 lists all paired tests. In each cell, the first number is the RSR difference between Source-MSIS and the comparison method in percentage points; the second is the adjusted p-value.

Table 9: Per-case paired tests between Source-MSIS and the five comparison methods.

Environment	Model	Who&When	AgenTracer	AgentDebug	CausalFlow	DoVer
Habitat	3B	$+5.0$ 0.875	$+2.5$ 1.000	-2.5 1.000	$+6.3$ 0.312	-6.3 0.875
	7B	$+48.8$ 1.82×10^{-11}	$+46.3$ 3.00×10^{-11}	$+51.3$ 3.00×10^{-11}	$+48.8$ 1.82×10^{-11}	$+46.3$ 1.46×10^{-10}
	12B	$+26.3$ 0.00383	$+63.8$ 3.42×10^{-13}	$+13.8$ 0.108	$+67.5$ 5.55×10^{-16}	$+41.3$ 1.63×10^{-6}
ScienceWorld	3B	$+51.3$ 6.62×10^{-23}	$+51.3$ 6.62×10^{-23}	$+51.3$ 6.62×10^{-23}	$+51.3$ 6.62×10^{-23}	$+51.3$ 6.62×10^{-23}
	7B	$+51.3$ 7.94×10^{-22}	$+52.0$ 3.31×10^{-23}	$+51.3$ 7.94×10^{-22}	$+52.0$ 3.31×10^{-23}	$+51.3$ 7.94×10^{-22}
	12B	$+52.7$ 1.65×10^{-23}	$+52.7$ 1.65×10^{-23}	$+52.7$ 1.65×10^{-23}	$+52.7$ 1.65×10^{-23}	$+52.7$ 1.65×10^{-23}
ALFWorld	3B	$+100.0$ 7.70×10^{-33}	$+100.0$ 7.70×10^{-33}	$+100.0$ 7.70×10^{-33}	$+100.0$ 7.70×10^{-33}	$+100.0$ 7.70×10^{-33}
	7B	$+99.1$ 1.23×10^{-32}	$+99.1$ 1.23×10^{-32}	$+99.1$ 1.23×10^{-32}	$+100.0$ 7.70×10^{-33}	$+98.2$ 1.23×10^{-32}
	12B	$+95.5$ 7.40×10^{-32}	$+98.2$ 2.47×10^{-32}	$+96.4$ 7.40×10^{-32}	$+99.1$ 1.54×10^{-32}	$+96.4$ 7.40×10^{-32}

Table 10 reports intervals for the differences between Source-MSIS and the comparison method with the highest RSR in each group. Intervals use 10,000 bootstrap resamples of paired tasks with seed 20260925. Ties follow the method order in Table 1. Habitat intervals for 3B and 12B cross zero, consistent with the paired tests. Differences remain positive for Habitat with 7B and for all model sizes in ScienceWorld and ALFWorld. The degenerate ALFWorld 3B interval reflects all observed pairs favoring Source-MSIS; it does not imply zero population uncertainty.

Table 10: RSR differences between Source-MSIS and the best comparison method in each group, with 95% paired bootstrap intervals.

Environment	Model	Comparator	RSR difference	95% interval
Habitat	3B	DoVer	-6.3	$[-15.0,\ 1.3]$
Habitat	7B	AgenTracer	$+46.3$	$[35.0,\ 57.5]$
Habitat	12B	AgentDebug	$+13.8$	$[-1.3,\ 28.8]$
ScienceWorld	3B	Who&When	$+51.3$	$[43.3,\ 59.3]$
ScienceWorld	7B	Who&When	$+51.3$	$[43.3,\ 59.3]$
ScienceWorld	12B	Who&When	$+52.7$	$[44.7,\ 60.7]$
ALFWorld	3B	Who&When	$+100.0$	$[100.0,\ 100.0]$
ALFWorld	7B	DoVer	$+98.2$	$[95.5,\ 100.0]$
ALFWorld	12B	Who&When	$+95.5$	$[90.9,\ 99.1]$

C Component and Controlled Analyses

This section supplements the component analysis in Section 4.3. We test historical source information with the same ALFWorld task solver under all three models. Separate 7B variants examine repair composition, dependency ordering, and residual feedback. Additional two-stage cases test feedback when the second source becomes visible only after the first repair. All comparisons use the same cases, action interfaces, and task evaluators within each experiment.

C.1 Historical Source Information with the Same Task Solver

Each of the 110 ALFWorld tasks is run independently with the full method and the control under 3B, 7B, and 12B. The two runs share the task parser, initial observation, exploration procedure, dependency constructor, action grammar, continuation policy, and numerical budgets. They use the same model checkpoint and seed stream, with run order randomized within each task and model pair using seed 20260925.

The control exposes only the task instruction, task type, initial observation, case identifier, and interaction budget to repair construction. Access to failed actions and historical observations is disabled. The same action-construction function is retained, while source claims and bindings are omitted when constructing the proposal. Both runs retain the original diagnostic model call and

continuation guidance; neither repair constructor receives the diagnostic text. No run receives the other run's proposal or feedback.

Table 11: ALFWorld outcomes with the same task solver. Each row contains 110 task pairs; the same tasks are evaluated under all three models.

Model	Full	Control	Identical plans	Identical resource counts
3B	110	110	110	110
7B	110	110	110	110
12B	110	110	110	110

All 660 executions complete successfully. Every paired proposal contains the same repair action sequence, and the control produces no source claims. Model calls, environment steps, environment replays, and evaluator calls also match within every pair. Wall-clock duration is not included in this equality comparison. These are repeated evaluations of the same 110 tasks, rather than 330 independent tasks.

C.2 Component Variants

Table 12: Recovery variants and their changes to the execution procedure.

Variant	Changed steps
Full Source-MSIS	Retains evidence-driven source error tracing, source–repair binding, multi-source composition, residual feedback, and dependency ordering.
Without historical source information	Retains the same task solver; repair construction cannot read historical failed actions or observations and emits no source claims or bindings.
Direct planning	Replaces the native repair solver with model-generated recovery actions from public failure records, with source records either omitted or retained as diagnostic context.
Single source error only	Retains only the highest-ranked candidate source, without accumulating a second candidate.
Without residual feedback	Plans once from the initial failed trajectory; subsequent selections do not read failure feedback after intervention execution.
Without order constraints	Retains the same candidate sources and repairs but does not use the execution order specified by task dependencies.

The original variants named "without source tracing" and "without source–repair binding" both use direct planning instead of the native ALFWorld solver. The former omits diagnostic source records; the latter retains them as context but does not use explicit bindings. Both complete 9 tasks with identical outcomes, and are grouped as direct planning in the following tables. The experiment in Appendix C.1 instead retains the native solver while changing its access to historical source information.

C.3 ALFWorld Recovery Variants with 7B

Table 13: ALFWorld recovery variants with 7B. Direct planning changes the solver; the remaining variants change options within the native solver.

Setting	Successful cases	RSR	Difference from full
Full Source-MSIS	**110/110**	**100.0%**	–
Direct planning	9/110	8.2%	−91.8
Single source error only	5/110	4.5%	−95.5
Without order constraints	23/110	20.9%	−79.1
Without residual feedback	110/110	100.0%	0.0

For each variant in Table 13, we compare its outcome with the full method on the same case. Table 14 reports paired outcomes and two-sided exact McNemar tests, with Holm correction for the original five comparisons. The recovery differences arise entirely from cases solved only by the full method. Removing residual feedback changes no ALFWorld case outcomes.

Table 14: Paired comparisons between the original ALFWorld recovery variants and full Source-MSIS.

Variant	Full only	Variant only	RSR difference	Adjusted p
Direct planning	101	0	+91.8	3.16×10^{-30}
Multi-source composition	105	0	+95.5	2.47×10^{-31}
Dependency ordering	87	0	+79.1	2.58×10^{-26}
Residual feedback	0	0	0.0	1.00

The original correction family contains both direct-planning variants, single-source repair, removal of dependency ordering, and removal of residual feedback. Both direct-planning comparisons remain in that family even though their identical outcomes occupy one displayed row. The new experiment with the same solver has no discordant outcomes under any model.

C.4 Residual Source Errors Revealed Only After Repair

Multiple errors in natural tasks sometimes appear together in the initial trajectory, allowing a single analysis to propose all candidates. To isolate the effect of residual feedback, we use 55 controlled cases with two source errors. In 28 cases, both errors are observable in the initial trajectory. In the other 27, evidence of the second error appears only after the first repair takes effect and execution continues. All methods use the same numbers of proposals and executions and the same testing rules.

Table 15: Successful case counts in the controlled residual source error experiment.

Case type	Source-MSIS	No residual feedback	Static search	Oracle
Both errors initially visible	28/28	28/28	28/28	28/28
Second error revealed after repair	**27/27**	0/27	0/27	27/27

When both errors are visible from the start, the full method, the variant without residual feedback, and static search all complete 28 cases. When the second error becomes observable only after the first repair, Source-MSIS completes all 27 cases, while neither of the other two settings completes any. The trajectory after the first repair supplies previously unavailable evidence for the second error; continuing to inspect this trajectory allows the method to add the next repair.

On the 27 cases with delayed exposure of the second error, the paired recovery success rate difference between the full method and the variant without residual feedback is 100 percentage points, with a two-sided exact McNemar test of $p = 1.49 \times 10^{-8}$. The difference is 0 when both errors are initially visible. Fisher's exact test for the difference between the two case types gives $p = 2.62 \times 10^{-16}$. By environment, the results are 20/20 versus 0/20 in MiniGrid and 7/7 versus 0/7 in ScienceWorld.

C.5 Controlled Source Error Localization and Certificate Checks

MiniGrid and synthetic ScienceWorld provide source errors, repairs, and task dependencies that can be checked individually. The controlled set contains 80 MiniGrid cases and 30 synthetic ScienceWorld cases, with 55 single-source and 55 two-source cases. Source-MSIS restores the task in all 110 cases. All resulting programs pass the proper order-preserving subsequence tests, and the identified source error sets match those recorded for the controlled cases. Table 16 counts the four outputs defined in Appendix A.4, using the saved checker results rather than task success alone.

Table 16: Source-MSIS verifier outcomes on controlled cases. The four output counts in each row sum to the number of cases.

Environment	Cases	Certified	Non-Minimal	Blocked	Unresolved
MiniGrid	80	80	0	0	0
Synthetic ScienceWorld	30	30	0	0	0
Total	110	110	0	0	0

For each single-repair sequence, the checker tested the empty sequence; for each two-repair sequence, it tested the empty sequence and both one-repair sequences. All 220 proper-subsequence replays failed to complete the task, while all 110 full-sequence replays succeeded. These results account for the CERTIFIED counts in the table; the other three outputs did not occur in this controlled set.

These controlled cases check whether candidate construction, source–repair binding, and certificate evaluation operate under the prescribed conditions. RSR in the main experiments on Habitat, natural ScienceWorld failures, and ALFWorld comes directly from the respective environment task evaluators. Manual source error labels are used only for the controlled cases above.

D REPEATED RECOVERY EXPERIMENT

We select 30 tasks per environment from the main experiment by task type and original failed trajectory length, without using methods' recovery results for selection. Each task is run independently five times with three models and the original five methods: Source-MSIS, Who&When, AgenTracer, AgentDebug, and CausalFlow. DoVer is included only in the main experiment. Generation temperature is 0.2, and the five generation seeds are 1729, 3253, 4789, 6421, and 7919. Initial environment states, environment seeds, prompts, execution budgets, and success criteria remain unchanged. Each run has separate execution records and does not reuse earlier generated responses or recovery results. Within each run, the original method's recovery procedure is retained. The 90 distinct tasks form 270 task and model combinations, with outcomes recorded separately for each model.

After the first 150 pilot executions, the experiment is expanded to 6750 executions, using each pilot record once and conducting 6600 additional executions. This expansion is an additional analysis initiated after inspecting the pilot results. Table 17 reports all outcomes, grouped into success in all five runs, a mixture of successes and failures, and failure in all five runs. Mean RSR uses 1350 executions per method, or 450 for each environment.

Table 17: Task completion across repeated runs. The upper block compares methods; the lower block separates Source-MSIS results by environment.

Method or environment	Combinations	Mean RSR	All five succeed	Mixed	All five fail
Who&When	270	7.3%	6	30	234
AgenTracer	270	2.1%	1	11	258
AgentDebug	270	8.2%	4	33	233
CausalFlow	270	1.1%	1	6	263
Source-MSIS	270	65.3%	151	49	70
Source-MSIS by environment					
Habitat	90	43.8%	14	49	27
ScienceWorld	90	52.2%	47	0	43
ALFWorld	90	100.0%	90	0	0

Source-MSIS succeeds in all five runs for every combination in ALFWorld. In ScienceWorld, 47 combinations succeed in all five runs and the other 43 fail in all five. Habitat shows more variation between runs: Source-MSIS achieves mean RSRs of 8.0%, 66.7%, and 56.7% with 3B, 7B, and 12B, respectively, compared with 10.7%, 10.7%, and 52.0% for the best comparison method in each configuration. The size of the recovery gap therefore still depends on the environment and model configuration.

Model generation plays different roles in the environment implementations. All 450 Source-MSIS executions in ALFWorld make only one initial diagnostic call. The diagnostic text is recorded separately and is not passed directly to the subsequent dependency constructor and repair solver. In ScienceWorld, later binding and planning steps still call the model. These execution paths identify which steps are affected when generation seeds change; the repeated outcomes measure recovery by each complete implementation.

E VERIFICATION OUTCOMES AND COSTS

E.1 CLASSIFICATION RECORDS AND COVERAGE

The classification index contains all 6,120 runs underlying Table 1: 340 tasks evaluated by six methods under three models. Recovery outcomes and verification categories are recorded separately for each run. This section reports the audit snapshot of September 25, 2026.

The audit applies the rules in Appendix A.4 to saved search termination records and completed independent replay checks. CERTIFIED and NON-MINIMAL require successful binding checks and full-sequence recovery, followed by the relevant deletion tests. A saved search that ends without recovery contributes to UNRESOLVED; these records do not require deletion tests. The categories below are assigned by the audit, rather than necessarily being outputs emitted by each baseline itself.

Table 18: Recorded categories across all main-experiment runs. Each method has 1,020 records. Unassigned indicates incomplete audit coverage, not a fifth verifier outcome.

Method	Certified	Non-Minimal	Blocked	Unresolved	Unassigned
Who&When	0	0	0	975	45
AgenTracer	0	0	0	1005	15
AgentDebug	0	0	0	962	58
CausalFlow	0	0	0	1015	5
DoVer	0	0	0	976	44
Source-MSIS	508	52	0	354	106
Total	508	52	0	5287	273

The index assigns 5,847 records to the four categories. The remaining 273 records all recovered in the main experiment. They comprise 167 baseline runs without independent verification outcomes and 106 Source-MSIS runs awaiting complete verification inputs. Of the latter, 102 Habitat plans lack a complete partition of the final rewritten plan into declared repairs. Four ScienceWorld plans append planner actions beyond the bound repair prefix. These records remain unassigned rather than being inferred from task success or counted as BLOCKED.

E.2 OUTCOMES BY ENVIRONMENT AND MODEL

Table 19–Table 21 expand the counts by environment and model. Each C/N/B/U entry gives CERTIFIED, NON-MINIMAL, BLOCKED, and UNRESOLVED, in that order. The last column gives unassigned counts for 3B, 7B, and 12B. The accompanying data provide the category and audit status for every individual run.

Table 19: Habitat classifications. Each method and model has 80 records.

Method	3B: C/N/B/U	7B: C/N/B/U	12B: C/N/B/U	Unassigned
Who&When	0/0/0/79	0/0/0/76	0/0/0/47	1/4/33
AgenTracer	0/0/0/77	0/0/0/74	0/0/0/77	3/6/3
AgentDebug	0/0/0/73	0/0/0/78	0/0/0/37	7/2/43
CausalFlow	0/0/0/80	0/0/0/76	0/0/0/80	0/4/0
DoVer	0/0/0/70	0/0/0/74	0/0/0/59	10/6/21
Source-MSIS	0/0/0/75	0/0/0/37	0/0/0/26	5/43/54

Table 20: ScienceWorld classifications. Each method and model has 150 records.

Method	3B: C/N/B/U	7B: C/N/B/U	12B: C/N/B/U	Unassigned
Who&When	0/0/0/150	0/0/0/149	0/0/0/150	0/1/0
AgenTracer	0/0/0/150	0/0/0/150	0/0/0/150	0/0/0
AgentDebug	0/0/0/150	0/0/0/149	0/0/0/150	0/1/0
CausalFlow	0/0/0/150	0/0/0/150	0/0/0/150	0/0/0
DoVer	0/0/0/150	0/0/0/149	0/0/0/150	0/1/0
Source-MSIS	75/0/0/73	76/1/0/72	78/0/0/71	2/1/1

Table 21: ALFWorld classifications. Each method and model has 110 records.

Method	3B: C/N/B/U	7B: C/N/B/U	12B: C/N/B/U	Unassigned
Who&When	0/0/0/110	0/0/0/109	0/0/0/105	0/1/5
AgenTracer	0/0/0/110	0/0/0/109	0/0/0/108	0/1/2
AgentDebug	0/0/0/110	0/0/0/109	0/0/0/106	0/1/4
CausalFlow	0/0/0/110	0/0/0/110	0/0/0/109	0/0/1
DoVer	0/0/0/110	0/0/0/108	0/0/0/106	0/2/4
Source-MSIS	108/2/0/0	91/19/0/0	80/30/0/0	0/0/0

Among the 330 Source-MSIS ALFWorld runs, 279 are CERTIFIED and 51 are NON-MINIMAL. Although every run recovered the task, deletion tests find redundant repairs in 2, 19, and 30 sequences under 3B, 7B, and 12B, respectively. ScienceWorld contributes 229 certified sequences and one non-minimal sequence. Together, these results distinguish task recovery from the necessity of every declared repair within a recovered sequence.

Habitat contributes no completed independent certificates in this snapshot. New runs that declare a complete repair partition are recorded separately from the original proposals. Their verification outcomes do not replace the original main-table recovery results or retrospectively certify an earlier unpartitioned plan.

E.3 REPAIR COUNTS AND VERIFICATION COSTS

The cost records cover 330 ALFWorld and 233 ScienceWorld verification runs, including three rejected full sequences. Verification executions and model calls are accounted for separately from the recovery search, following Appendix A.6. Table 22 reports all available records, not only certified cases.

Table 22: Observed verification costs. Executions, environment steps, and model calls are means per recorded run; the last column gives the maximum physical executions.

Environment	Model	Records	Executions	Steps	Model calls	Max.
ALFWorld	3B	110	7.22	288.98	263.87	9
	7B	110	7.22	321.68	295.95	9
	12B	110	7.22	288.36	263.08	9
ScienceWorld	3B	78	1.96	69.31	52.73	2
	7B	77	2.00	65.66	48.95	2
	12B	78	2.00	67.67	51.09	2

Under each ALFWorld model, 49 sequences contain two declared repairs and 61 contain three, giving a mean of 2.55 repairs. Complete outcome testing therefore requires four or eight replays, averaging 6.22 per sequence. An additional execution inspects local state transitions, bringing physical executions to five or nine, with a mean of 7.22. Every recorded ScienceWorld sequence contains one declared repair; complete outcome testing requires two replays. The three rejected full sequences stop before the empty-sequence test.