Repair or Resample? Rethinking Failure Debugging in LLM Multi-Agent Systems

Zhongwen Luan[1], Xiaoyu Zhang[2], Ming Hu[1,3],
Yue Yang[4], Jiongchi Yu[2,3], Xiaohong Chen[1]

[1]East China Normal University
[2]Nanyang Technological University
[3]Singapore Management University
[4]Xi'an University of Architecture and Technology

10245101408@stu.ecnu.edu.cn

Abstract

As large language model (LLM)-based multi-agent systems (MASs) are increasingly applied to long-horizon complex tasks, their reliability has emerged as the core bottleneck hindering their real-world deployment. Existing MAS debugging and repair methods typically rely on rerunning and resampling the entire execution trajectory. However, a fundamental question remains to be answered: *do these methods causally repair MAS failures or merely stochastically repair by leveraging the randomness of LLM sampling?* To evaluate the effectiveness of MAS repair methods, we introduce SymTrace, a controlled evaluation framework that records the MAS execution trajectory and establishes intervention anchors. During replay, it effectively reconstructs the execution before the anchor using recorded logs and only regenerates the downstream trajectory, thereby enabling the reliable reproduction of MAS failures. We further construct the dataset SymFail, comprising 536 human-annotated failure trajectories with graph-linked locations, categories, and trace evidence. Based on these foundations, we conduct a large-scale empirical study across three mainstream MAS frameworks. Our findings reveal that existing unguided rerun methods are highly unreliable, exhibiting low failure reproduction and repair rates (only 67.97% and 6.90%, respectively). Building upon these findings, we further explore the effectiveness of a symptom-driven intervention method, which successfully repairs 20.15% of the failed cases (a 191.89% improvement to state-of-the-art repair methods). This study aims to provide actionable insights for MAS debugging and repair research, paving the way for the robust deployment of multi-agent systems.

1 Introduction

Large language model (LLM)-based multi-agent systems (MASs) coordinate specialized agents through message passing, shared context, and structured workflows (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Fourney et al., 2024). By distributing planning, information gathering, tool use, execution, and verification, MASs offer a promising foundation for long-horizon applications such as web interaction and software development (Zhou et al., 2024; Yoran et al., 2024; Yang et al., 2024). As these systems are deployed in increasingly complex and consequential settings, systematically understanding, reproducing, and repairing their failures becomes essential to reliability and safety.

Existing research has advanced MAS reliability through benchmarks, failure diagnosis, and execution repair. Benchmarking and diagnostic studies collect execution trajectories, construct failure taxonomies, and localize responsible agents or steps (Yao et al., 2024; Ye et al., 2026; Cemri et al., 2025; Shah et al., 2026; Deshpande et al., 2025; Zhang et al., 2025), while repair methods use rerunning, reflection, critic feedback, or trajectory-level guidance to generate new executions (Madaan et al., 2023; Shinn et al., 2023; Du et al., 2023; Gou et al., 2024; Zhao et al., 2026b; Nanda et al., 2026; Zhao et al., 2026a). However, two limitations remain. First, complete reruns resample upstream model decisions instead of holding the failure-producing execution fixed, making terminal success difficult to attribute to the applied repair. Second, task-level verdicts and automatically assigned categories may not identify the trace-localized behavior that actually requires intervention. Figure 1 illustrates how both the diagnosis and ob-

Source Code: https://anonymous.4open.science/r/SymTrace-7234

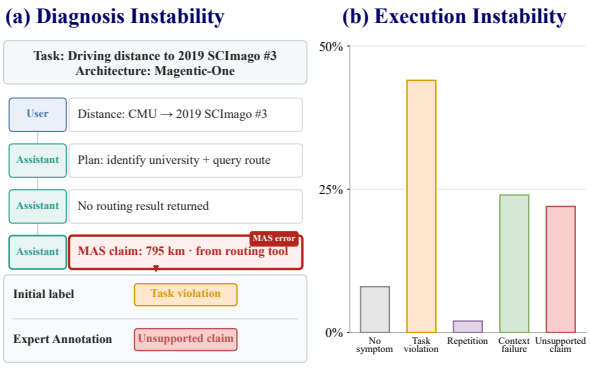


Figure 1: Diagnosis and execution instability for the same MAS task. (a) In a Magentic-One distance-query failure, the system makes a routing-derived claim despite receiving no routing result, leading to disagreement between the initial and expert diagnoses. (b) Repeated stochastic reruns of the same task produce different failure types, while some exhibit no detected symptom.

served outcome can change without establishing that the recorded failure was corrected. A central question therefore remains: *Do these methods causally repair MAS failures, or merely achieve stochastic recovery through LLM resampling?*

To fill this gap, we introduce SymTrace, a replay-oriented logging system, together with SymFail, a human-annotated failure-trajectory dataset. During snapshot recording, SymTrace captures model and tool boundary interactions, organizes them into an event-dependency graph and observed execution order, and materializes the resulting records as a replay bundle. During replay, it restarts the native MAS, strictly matches each intercepted request, and injects the corresponding recorded result until reaching the designated intervention boundary. Live execution then resumes with the repair applied. This procedure holds the observable pre-intervention history fixed so that downstream changes can be attributed to the intervention rather than upstream resampling. Complementing this execution control, SymFail is constructed from 200 WebArena-Verified Hard and AssistantBench tasks (Zhou et al., 2024; Yoran et al., 2024) executed with AG2 (Wu et al., 2023), CrewAI (CrewAI, Inc., 2026), and Magentic-One (Fourney et al., 2024). Of the resulting 600 trajectories, 536 evaluator-confirmed failures are retained and human-annotated with graph-linked failure nodes, symptom categories, supporting evidence, and final evidence-based annotations.

Using these two resources, we evaluate task-level repair and node-level controls across 536 failures. SymTrace increases per-execution failure reproduction from 67.97% to 80.78% and consistent reproduction across three executions from 41.42% to 52.43%. Task-level regeneration repairs at most 6.90% of failures within three attempts, whereas *Suspicious-Node Intervention* repairs 20.15% with a single intervention (2.92× the rate of the strongest task-level baseline) and performs best across all three MASs. These results demonstrate the value of controlled execution and localized evidence over repeated stochastic regeneration.

Our contributions include: ❶ We design and implement SymTrace, a logging system that records each MAS execution as an event-dependency snapshot and supports verifiable prefix reconstruction and replay, enabling intervention without resampling the preceding execution. ❷ We construct SymFail, a human-annotated dataset including 536 human-annotated failure trajectories from WebArena-Verified Hard and AssistantBench, which forms the foundation of the MAS debugging study. ❸ We conduct a large-scale study across three MASs measuring how reliably existing methods reproduce and repair recorded failures. Guided by our findings, we propose a symptom-driven intervention method that localizes replayable anchors and applies evidence-conditioned repair, improving over the strongest task-level baseline by 191.89%. ❹ We release SymTrace, SymFail, and our experimental results to support reproducibility and future MAS debugging and repair research.

2 Background & Related Work

2.1 LLM-Based Multi-Agent Systems

LLM-based multi-agent systems (MASs) coordinate multiple LLM-powered agents to accomplish a shared task. Each agent is typically assigned a role, instructions, context, and a set of tools, while a concrete execution proceeds through model calls, inter-agent messages, intermediate artifacts, and interactions with external environments. The coordination architecture determines how tasks are decomposed, which agent acts next, and how intermediate state is transferred. The systems evaluated in our study cover three representative designs. AG2 supports programmable conversational interactions among customizable agents (Wu et al., 2023); CrewAI organizes role-specialized agents through sequential or hierarchical task processes (CrewAI, Inc., 2026); and Magentic-One uses a central Orchestrator to plan, delegate tasks to specialized agents, track progress, and initiate replanning (Fourney et al., 2024). These systems exemplify conversation-centric, workflow-based, and orchestrator-centered coordination, re-

spectively, but all produce dependency-structured trajectories in which earlier decisions shape subsequent messages, actions, and observations. Consequently, MAS debugging and repair must examine how failures emerge and propagate through the execution trajectory rather than relying solely on the terminal output.

2.2 MAS Debugging and Repair

MAS debugging aims to make multi-agent executions inspectable by identifying failure types, responsible components, and error locations. Existing work develops structured observability mechanisms, failure taxonomies, annotated traces, and localization benchmarks for agentic workflows (Dong et al., 2024; AlSayyad et al., 2026; Shah et al., 2026; Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2025). These studies provide foundations for describing and locating failures, but their traces are primarily used for post-hoc analysis. In contrast, SymTrace records model and tool interactions together with their dependencies, execution order, and realized results, turning execution traces into replayable records that support verifiable reconstruction.

MAS repair commonly regenerates failed behavior using self-generated feedback, peer critique, tool-grounded evidence, runtime diagnosis, or dependency-aware localization (Madaan et al., 2023; Shinn et al., 2023; Du et al., 2023; Gou et al., 2024; Zhang et al., 2026; Zhao et al., 2026b; Nanda et al., 2026; Zhao et al., 2026a). However, these methods are generally evaluated by terminal success without consistently controlling the execution before repair. Consequently, a successful rerun may repair the recorded failure or merely avoid it through stochastic resampling. Building on execution replay for controlling nondeterminism (Ronsse et al., 2000), SymTrace reconstructs the observed execution before an evidence-supported intervention, enabling our study to distinguish targeted repair from stochastic repair.

3 System Design

Figure 2 presents the two-mode workflow of SymTrace. The pipeline takes a native MAS execution, including its task, represented initial state, runtime configuration, and realized model and tool interactions, as input. Snapshot Mode processes this execution through Boundary Logging and Trace Construction, and then uses Replay Bundle to materialize the recorded trajectory. Replay Mode takes this trajectory together with an intervention target as input, reconstructs the target prefix through Result Injection and Boundary Matching, and applies the intervention through Live Resume. Replay Scope finally specifies the guarantees attached to the reconstructed prefix and the newly generated suffix. The overall output is a new MAS trajectory that preserves the validated execution before the target and may diverge from the source trajectory from the intervention onward.

3.1 Snapshot Mode

Snapshot Mode transforms a complete native MAS execution into a structured trajectory that contains the information required for subsequent replay. Specifically, Boundary Logging records the realized model and tool interactions, Trace Construction organizes the recorded events into their dependency structure and observed order, and Replay Bundle combines these results with the task and runtime configuration. The resulting bundle is the output of Snapshot Mode and the trajectory input to Replay Mode.

Boundary Logging. During the native execution, framework-specific hooks intercept exposed LLM request-response pairs and tool call-observation pairs without changing the MAS scheduler, agent logic, or state-update procedures. Each request continues to its live endpoint, while SymTrace records the realized request, result, and event position. This process converts the runtime interactions into ordered boundary records R, which preserve the nondeterministic model and tool values needed for replay.

Trace Construction. The boundary records describe the values observed at individual model and tool interactions but do not capture how those interactions depend on one another. Trace Construction therefore organizes the realized events into an event-dependency graph $G = (V, E)$, whose edges represent data or control dependencies. Repeated workflow iterations are stored as distinct event instances, and the observed event order $O = (v_1, \ldots, v_n)$ is recorded separately because the graph may permit multiple valid schedules. The resulting G and O provide the structural information associated with the boundary values in R.

Replay Bundle. Replay Bundle combines the recorded values and execution structure with the information required to restart the native MAS:

$$\mathcal{S} = \langle T, s_0, c, G, O, R \rangle , \tag{1}$$

where T is the task, s_0 is the represented initial state, and c is the runtime configuration. The resulting bun-

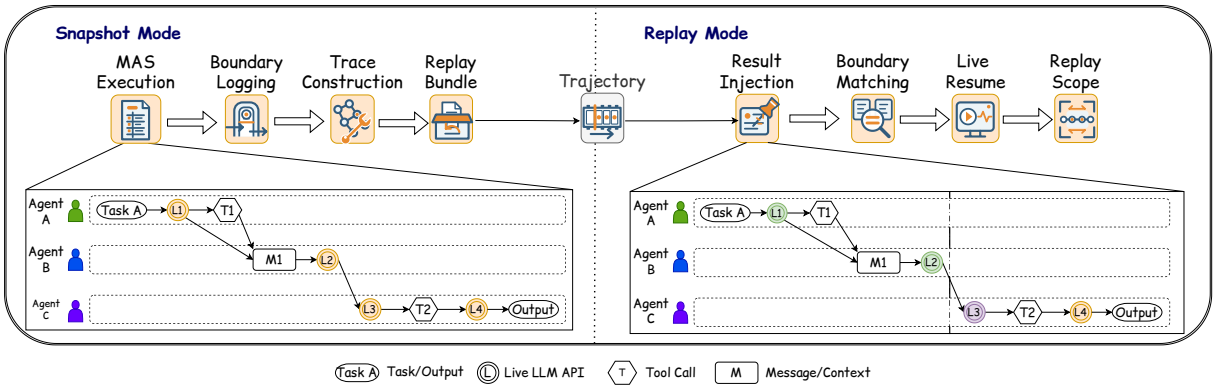


Figure 2: Overview of the SymTrace workflow. (Snapshot Mode transforms a native MAS execution into a recorded trajectory through Boundary Logging, Trace Construction, and Replay Bundle. Replay Mode consumes the trajectory, reconstructs the target prefix through Result Injection and Boundary Matching, applies the intervention through Live Resume, and specifies the resulting guarantee through Replay Scope. The lower panels provide an illustrative source and replayed execution.)

dle $\mathcal{S}$ materializes the recorded trajectory transferred from Snapshot Mode to Replay Mode.

3.2 Replay Mode

Replay Mode reconstructs the recorded execution before a designated target and resumes the MAS from that point under an intervention. Specifically, Result Injection associates each intercepted prefix call with its historical boundary result, Boundary Matching verifies that the current call corresponds to the expected record, and Live Resume applies the intervention and restores live execution. Replay Scope then characterizes which part of the resulting trajectory is guaranteed by replay. We refer to this procedure as selective replay because only the recorded prefix before the target is reconstructed, while execution from the target onward remains live.

Result Injection. Replay Mode begins by restarting the native MAS using T, s_0, and c from the replay bundle. Whenever the restarted MAS reaches a model or tool boundary before the target, Result Injection retrieves the corresponding historical record from R instead of immediately invoking the live endpoint. The intercepted call and candidate record are then supplied to Boundary Matching, so the recorded result is returned only after the correspondence has been verified.

Boundary Matching. Boundary Matching validates the intercepted call against the expected record using its event position and canonicalized request content. A mismatch terminates replay, whereas a successful match authorizes the recorded result to be returned to the native MAS. Processing this result advances the MAS to its next represented state and produces the next boundary call. Repeating this process according to G and O reconstructs the target prefix, while content hashes validate the reused prefix nodes.

Live Resume. Once the verified prefix reaches the designated target, Live Resume stops returning historical boundary results and applies the intervention. Model and tool interactions are restored to their live endpoints, while the native scheduler and state-update procedures continue unchanged. This stage uses the reconstructed prefix and intervention as its starting state and produces a newly generated downstream trajectory.

Replay Scope. Replay Scope defines which parts of the resulting trajectory are reconstructed from recorded results and which parts remain live. Before the target, every reused boundary result is returned only after strict matching, preserving the represented logical history of the recorded execution. From the intervention onward, model generation, tool responses, and downstream decisions remain live and may differ from the source trajectory. This guarantee holds when native transitions are deterministic under the recorded boundary values and relevant external state is reset, isolated, or restored. It does not require equality of inaccessible framework internals or deterministic behavior in the new suffix.

4 Dataset Construction

Based on WebArena-Verified Hard (Zhou et al., 2024) and AssistantBench (Yoran et al., 2024), we construct and annotate SymFail, comprising 536 evaluator-confirmed failure trajectories with structured execution traces, event graphs, localized failure nodes, and multi-label annotations.

4.1 Candidate Selection

The candidate set comprises all 258 WebArena-Verified Hard tasks and the 33 eligible AssistantBench development tasks with an available task description, reference answer, and gold URL. We retain all 33 AssistantBench tasks and the first 167 WebArena-Verified Hard tasks in their published order, producing a deterministic pool of 200 tasks before any MAS execution or repair experiment. The final pool contains 33 Web-QA, 70 mutation, 25 navigation, and 72 retrieval tasks. We select these benchmarks because they provide multi-step Web tasks with externally verifiable outcomes.

4.2 Trajectory Collection

Each task is executed once with AG2 (Wu et al., 2023), CrewAI (CrewAI, Inc., 2026), and Magentic-One (Fourney et al., 2024), producing $200 \times 3 = 600$ initial task-MAS executions. The native benchmark evaluators identify 536 failures with complete trace and graph artifacts, while the remaining 64 executions are excluded before annotation. The 501 WebArena-Verified executions contribute 462 failures and the 99 AssistantBench executions contribute 74 failures. The retained trajectories comprise 171 AG2, 184 CrewAI, and 181 Magentic-One executions.

4.3 Failure Annotation

We derive C1-C4 from the Multi-Agent System Failure Taxonomy (Cemri et al., 2025), which identifies 14 fine-grained failure modes under three broader categories. We retain the four most prevalent consolidated patterns in the source analysis, producing the non-exclusive, trace-observable categories in Table 1.

All four annotators have software-engineering backgrounds and experience in software development and code analysis. Before the main annotation, they calibrate on reference-labeled examples and practice trajectories using shared category, evidence, and node-selection guidance. Three annotators then independently assign all applicable categories, select one primary category, and identify the earliest trace-supported actionable node with evidence and a rationale. A fourth annotator reviews the task, complete trace, event graph, and initial annotations to produce the final annotation, with authority to revise any decision rather than applying majority vote or label union.

The final category set differs from the initial label union in 113 of 536 trajectories (21.08%). Agreement is measured only among the three independent annotators: Fleiss' κ is 0.62 for the primary category and 0.81 for failure-node type, while exact node agreement is 73.88% among all three annotators and 95.90% for at least two annotators. All $536 \times (3+1) = 2{,}144$ initial and final node selections exist in their source event graphs and match their recorded node types. The release includes the task manifest, linked traces and graphs, initial and final annotations, rationales, and supporting evidence. Appendix B provides the complete annotation guidance and reliability analysis.

Table 1: Diagnosis-to-repair mapping. (Categories represent trace-localized intervention signals rather than task-level failure verdicts.)

Category	Repair signal
C_1	Implicated constraint and conflicting behavior; request a compliant alternative.
C_2	Repeated history and unfinished objective; request an action that makes progress.
C_3	Unresolved runtime condition and its evidence; require resolution before continuing.
C_4	Explicit plan, action, and outcome; request a decision that resolves their inconsistency.

5 Study

5.1 Setup

Study scope. The primary evaluation applies AG2 (Wu et al., 2023), CrewAI (CA) (CrewAI, Inc., 2026), and Magentic-One (MO) (Fourney et al., 2024) to the 200 fixed tasks used to construct SymFail, yielding 536 source failures from 600 runs (Section 4). Each source failure is one experimental unit; executions of the same task by different MASs are treated separately. The evaluation set is frozen before replay and repair.

Baselines. We compare three representative task-level debugging and repair methods. For each source failure, all methods use the same task input, MAS, model alias, temperature, and native evaluator. ❶ *Unguided Full Rerun* is a retry-based repair method that restarts the complete MAS execution from the original task without using information from the failed execution (Brown et al., 2024). ❷ *Self-Reflection* is a feedback-based repair method that provides the MAS with its failed final output and asks it to identify possible mistakes before solving the task again (Madaan et al., 2023; Shinn et al., 2023).

❸ *Critic-Agent* is a critic-based repair method that employs a separate critic agent to review the failed output and provide corrective guidance for a new complete execution (Gou et al., 2024). Each task-level method receives up to three complete attempts and stops after its first success, whereas each node-level method receives one selective-replay intervention.

Evaluation metrics. Following τ-bench (Yao et al., 2024) and the *pass@k* convention (Chen et al., 2021), we define two metrics parameterized by the attempt count k. Let N denote the number of source cases, $z_{i,a}^m$ indicate whether method m reproduces the source failure for case i in attempt a, and $y_{i,a}^m$ indicate whether the native evaluator accepts the corresponding repair.

- **Failure reproduction** (rep_k). Given three executions per case, let $\mathcal{A}_k = \{S \subseteq \{1,2,3\} : |S| = k\}$ denote all subsets of k executions:

$$\text{rep}_k(m) = \frac{1}{N|\mathcal{A}_k|} \sum_{i=1}^{N} \sum_{S \in \mathcal{A}_k} \prod_{a \in S} z_{i,a}^m. \quad (2)$$

  This metric describes the proportion of execution subsets in which all k executions reproduce the source failure.

- **Repair success** (pass@k):

$$\text{pass@}k(m) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left[\max_{a \in \{1,\ldots,k\}} y_{i,a}^m = 1\right]. \quad (3)$$

  This metric describes the proportion of source failures that receive at least one evaluator-accepted repair within k attempts.

Model and efficiency reporting. All experimental conditions use `deepseek-v4-flash` (Xu et al., 2026) at temperature 0.00 through the same OpenAI-compatible endpoint. We report captured API calls and accepted repairs per 1,000 calls as secondary efficiency measures, compared primarily within each MAS.

Statistical analysis. Repair rates are reported with 95% Wilson confidence intervals to quantify uncertainty in the estimated proportions. Paired rate differences are reported with 95% confidence intervals obtained from 10,000 case-level bootstrap resamples, measuring the magnitude and uncertainty of the difference between methods. Two-sided exact McNemar tests compare paired binary outcomes on the same source cases, while Holm correction controls the family-wise error rate across prespecified multiple comparisons within each evaluation setting and MAS. Appendix C provides the complete execution and statistical conventions.

Table 2: Same-failure reproduction rates (%) by MAS on the 536 SymFail source failures. (CA and MO denote CrewAI and Magentic-One; Rerun and Replay denote Unguided Full Rerun and SymTrace Replay. Bold marks the higher value in each column.)

	rep_1 (%)				rep_3 (%)			
Method	**AG2**	**CA**	**MO**	**Total**	**AG2**	**CA**	**MO**	**Total**
Rerun	64.13	77.36	62.06	67.97	36.26	52.72	34.81	41.42
Replay	**80.70**	**81.88**	**79.74**	**80.78**	**50.29**	**53.26**	**53.59**	**52.43**

5.2 RQ1: How Reliably Can MAS Failures Be Reproduced?

Experimental design. To assess whether reconstructing the recorded execution history reproduces failures more reliably than resampling an entire execution, we analyze 536 source failures with three attempts per method, comparing SymTrace *Replay*, which reconstructs the recorded prefix and resumes at the target failure node, against *Unguided Full Rerun*, which restarts the original task without using any guidance. We use an LLM-as-a-judge pipeline adapted to our taxonomy from Cemri et al. (2025), with the SymFail labels as references. Table 2 reports the results, where the rows represent the two methods, the columns represent the three MASs and their aggregate, and the two column groups report rep_1 and rep_3, respectively.

Analysis of reproduction reliability. The reproduction results show that SymTrace achieves a consistent advantage over full rerun across all three MASs. All reproduced prefix nodes pass content-hash validation, yielding 100.00% prefix exactness. The remaining gap between exact prefix reconstruction and end-to-end reproduction can be attributed to short replay prefixes, post-target execution variation, and evaluation uncertainty.

First, most cases preserve only a short execution prefix. Specifically, 381 of the 536 cases (71.08%) reuse only 2-3 nodes, including 738 of the 1,608 attempts (45.90%) that reuse exactly two. Within this dominant group, the rep_1 and rep_3 gains over full rerun are 9.80 and 6.82 percentage points. The corresponding gains increase to 17.76 and 18.69 points for 4-8 reused nodes and to 25.69 and 27.08 points for at least nine nodes.

Second, SymTrace reuses only the nodes preceding the failure target. The target and its successors remain subject to live model and tool execution. For `webarena_verified_hard_127` on AG2, three attempts reconstruct identical prefixes but generate three different unsupported answers without intervening tool calls. For `webarena_verified_hard_267`, the source execution receives an HTTP 200 response

from the Wikipedia API, whereas replay receives an HTTP 429 response. Such executions may retain the same high-level failure category while exhibiting different case-specific failures and are therefore not counted as successful reproductions.

Third, LLM-as-a-judge errors introduce measurement uncertainty. In a stratified secondary audit of 72 judgments, 64 agree with the original labels, five are confirmed disagreements, and three remain uncertain. After weighting by stratum, the estimated confirmed-disagreement rate is 3.30%, the uncertainty rate is 3.67%, and their combined sensitivity upper bound is 6.96%. All five confirmed disagreements are false negatives in which a genuine reproduction was labeled as different_failure.

Finding 1. SymTrace *improves failure reproduction by exactly reconstructing the execution before the failure target. Although it cannot control model and environmental variation after that target, its advantage grows when more preceding steps can be reused, making it particularly suitable for realistic long-horizon MAS tasks.*

5.3 RQ2: Can Task-Level Re-execution Reliably Repair Failed MAS Executions?

Experimental design. To understand whether task-level repair methods correct source failures or mainly benefit from repeated sampling, we use SymFail as the evaluation basis and apply three baseline repair methods to its 536 failed executions from AG2, CrewAI, and Magentic-One. The results are presented in Table 3. Each row reports one subset of the 536 failures, grouped in the first column by MAS and by primary failure category (C1-C4), with its case count and the pass@3 repair rate of each baseline. Both groupings share the same overall total.

Analysis across evaluation settings. The results on SymFail show that generic reflection and critique do not provide a consistent repair advantage over unguided task restart. Unguided Full Rerun performs best for every MAS, while the category breakdown provides no evidence that the feedback-based methods systematically address particular failure types. This suggests that their additional feedback does not reliably target the source failure mechanism.

The comparison with the original CRITIC and Reflexion studies highlights two factors affecting reported repair rates: task difficulty and repeated sampling. SymFail consists of failed long-horizon Web executions involving multi-agent coordination, dynamic information acquisition, and multiple dependent actions, making successful regeneration particularly difficult. More importantly, both original studies report increasing success over successive trials (Gou et al., 2024; Shinn et al., 2023). Because each additional trial provides another opportunity to sample a different answer or trajectory, the resulting gains combine feedback effects with repeated regeneration. Under the same three-attempt budget on SymFail, the feedback-based methods still do not outperform unguided rerunning.

Table 3: Task-level pass@3 results (%) of the three task-level baselines on SymFail. (The first column groups the same 536 failures by MAS and by primary failure category (C1-C4). Both breakdowns share the single overall **Total**. 'Rerun', 'Self', and 'Critic' denote Unguided Full Rerun, Self-Reflection, and Critic-Agent.)

	Group	Cases	pass@3 (%)		
			Rerun	Self	Critic
MAS	AG2	171	8.19	4.68	4.09
	CrewAI	184	3.80	2.17	2.72
	Magentic-One	181	8.84	6.08	4.42
Category	C1	125	9.60	4.80	2.40
	C2	69	4.35	2.90	5.80
	C3	203	5.42	3.45	1.97
	C4	139	7.91	5.76	6.47
Total		**536**	**6.90**	**4.29**	**3.73**

Analysis of outcome instability. Complete re-execution changes outcomes in both directions, confirming the stochasticity of the task-level baselines and showing that an accepted rerun does not necessarily indicate repair of the source failure. To examine this behavior, we use 54 initially successful executions recorded by SymTrace during the same data-collection process and rerun each execution three times under the same runtime configuration. Among the resulting 162 attempts, 85 result in failures, and 39 of the 54 source cases regress at least once.

Case analysis of repair mechanisms. A CrewAI case shows that an evaluator-accepted answer can be produced without resolving the source failure mechanism. The task asks for the rate of snowy New Year's Eves in Chicago from 2014 to 2023, but the source execution and all three repair methods lack the required observations because the extracted dynamic pages do not contain them. Self-Reflection and Critic-Agent report that the requested rate cannot be determined, whereas Unguided Full Rerun returns the accepted answer of 30.00% while acknowledging that it relies on general historical patterns rather than the required evidence.

Finding 2. *Task-level methods repair primarily through stochastic regeneration rather than failure-*

specific repair. Re-execution can both repair failed outcomes and destabilize successful ones, while feedback-based variants provide no advantage under equal budgets. Thus, observed repair alone does not demonstrate that the source failure was identified and repaired.

5.4 RQ3: Are Node-Level Symptom Signals Actionable for Targeted Repair?

Experimental design. To evaluate whether trace-localized symptoms provide actionable evidence for targeted repair, we analyze the 536 source failures from AG2, CrewAI (CA), and Magentic-One (MO). We introduce *Suspicious-Node Intervention*, a node-level repair method that treats a trace node as a candidate intervention anchor when its local behavior exhibits observable failure symptoms connected to the unsuccessful outcome. During execution, deterministic rules and a semantic judge jointly evaluate each newly completed node for the C1-C4 symptoms defined in Table 1. When the judge assigns the current node a suspicion score above a predefined threshold, the controller suspends the ongoing MAS execution. Because the suspicious behavior has already occurred by the time it is detected, SymTrace reconstructs the execution prefix leading to that boundary, applies a repair instruction conditioned on the judge's C1-C4 classification, and then resumes live execution from the intervention point. We compare this method with Random-Node Intervention and Last-Node Intervention, which receive the same single selective-replay opportunity but no symptom evidence, and with the three task-level debugging and repair methods defined in Section 5.3.

Table 4 reports the budgeted pass rate of each method on each MAS and over all 536 failures. For the three node-level methods, the reported value is pass@1 because each method receives one selective-replay intervention. For the task-level methods, it is pass@3 because they receive up to three complete-execution attempts and stop after the first success. This comparison is conservative with respect to Suspicious-Node Intervention: the proposed method must repair the task through a single symptom-conditioned intervention, whereas the task-level baselines have three opportunities to obtain a successful outcome through complete re-execution and stochastic resampling. Thus, the larger pass rate of Suspicious-Node Intervention cannot be attributed to a larger attempt budget. Appendix F presents the complete pipeline, detection rules, threshold configuration, and intervention prompts.

Table 4: RQ3 repair utility on the 536 SymFail source failures. (AG2, CrewAI (CA), and Magentic-One (MO) contribute 171, 184, and 181 failures, respectively. 'Att.' is the attempt budget: node-level methods use one selective-replay intervention and task-level methods use up to three complete attempts. Bold marks the best result.)

Method	Att.	Pass rate (%)			
		AG2	CA	MO	Total
Last-Node	1	0.58	1.63	1.66	1.31
Random-Node	1	2.34	4.89	3.87	3.73
Critic-Agent	3	4.09	2.72	4.42	3.73
Self-Reflection	3	4.68	2.17	6.08	4.29
Unguided Full Rerun	3	8.19	3.80	8.84	6.90
Suspicious-Node	1	**16.37**	**25.00**	**18.78**	**20.15**

Analysis of repair effectiveness. Suspicious-Node Intervention achieves the highest overall repair rate and nearly triples the strongest task-level baseline, despite receiving only one intervention rather than three complete attempts. It also substantially outperforms Random-Node Intervention and Last-Node Intervention under the same selective-replay budget. The comparison with Random-Node Intervention and Last-Node Intervention shows that selective replay alone does not explain the improvement. Together, these results indicate that the improvement is not explained solely by additional sampling opportunities or access to an intervention point. Suspicious-Node Intervention additionally uses symptom-based target selection and repair guidance. Because target selection and repair guidance are evaluated jointly, the experiment does not attribute the full improvement to either component in isolation.

Analysis of cross-MAS consistency. Although absolute repair effectiveness varies across architectures, the method ranking remains stable: Suspicious-Node Intervention performs best on every MAS. This consistency is notable because the task-level baselines receive up to three opportunities to benefit from stochastic resampling, whereas Suspicious-Node Intervention uses only one symptom-guided intervention. It significantly outperforms Unguided Full Rerun on all three MASs after within-MAS Holm correction, showing that the overall advantage is not driven by a single architecture. Localized symptom evidence therefore provides a more consistent repair signal than repeated full-trajectory regeneration.

Finding 3. *Across MAS architectures, repair effectiveness depends more on precise execution control than on repeated regeneration. Localized symptoms provide actionable repair guidance, whereas full-trajectory resampling cannot reveal what was actually corrected.*

6 Discussion

This section discusses the limitations and potential future directions. ❶ **Data Coverage**. We ground SymFail in established, externally verifiable benchmarks (i.e., WebArena-Verified Hard, AssistantBench). This yields reproducible, outcome-verified failures, but benchmark-derived tasks cannot capture the full diversity of failures in deployed MASs. Therefore, collecting real-world failure trajectories and covering further MAS structures, interaction patterns, and task domains is a valuable future direction. ❷ **Automated Evaluation**. In the failure-reproduction experiment, we use LLM judges to determine whether a generated execution reproduces the source failure. The judges localize and classify the failure in the generated trajectory, compare these results with the reference location and category, and decide whether they represent the same failure. Although this automated evaluation enables large-scale repeated experiments, LLM judges may misinterpret long trajectories, ambiguous evidence, or semantically similar failure locations, and our manual spot checks identify occasional errors. Future work can incorporate expert verification for disagreements and low-confidence cases, develop hybrid human-LLM annotation, and report human-LLM agreement alongside automated results. ❸ **Model Coverage**. We use the same recent model alias and generation configuration across all MASs and repair methods. This controlled setting is necessary for our large-scale paired comparisons: it prevents differences in model capability or version from confounding differences between MAS architectures and repair strategies, while evaluating SymTrace with a model representative of current agent systems. Nevertheless, results from a single model may not generalize to other model families, providers, scales, or future versions. Future studies can repeat the evaluation across a broader set of models to test whether the observed reproduction and repair patterns remain consistent.

7 Conclusion

This study distinguishes correction of a recorded MAS failure from stochastic repair through a different execution. We introduce SymTrace as a controlled evaluation framework, construct SymFail with human annotation, and conduct a cross-system study of failure reproduction and repair. Across systems and tasks, failure identity depends on execution history rather than task input alone. Full task regeneration conflates repair with outcome variability, making terminal success insufficient evidence that the original mechanism was corrected. Localized symptoms can nevertheless guide effective intervention without uniquely identifying a root cause. Reliable repair evaluation should therefore preserve the history before intervention, localize the intended change, and assess whether the failure mechanism was corrected.

References

Adam AlSayyad, Kelvin Yuxiang Huang, and Richik Pal. Agenttrace: A structured logging framework for agent system observability. In *LLM-based Multi-Agent Systems: Towards Responsible, Reliable, and Scalable Agentic Systems*, 2026.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

CrewAI, Inc. CrewAI: Framework for orchestrating role-playing autonomous AI agents. https://github.com/crewAIInc/crewAI, 2026. GitHub repository, accessed July 25, 2026.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025.

Liming Dong, Qinghua Lu, and Liming Zhu. Agentops: Enabling observability of llm agents. *arXiv preprint arXiv:2411.05285*, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pages 57734–57811, 2024.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pages 23247–23275, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Rahul Nanda, Chandra Maddila, Smriti Jha, Euna Mehnaz Khan, Matteo Paltenghi, and Satish Chandra. Wink: Recovering from misbehaviors in coding agents. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software*, pages 208–217, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.

Michiel Ronsse, Koen De Bosschere, and Jacques Chassin de Kergommeaux. Execution replay and debugging. *arXiv preprint cs/0011006*, 2000.

Mehil B Shah, Mohammad Mehdi Morovati, Mohammad Masudur Rahman, and Foutse Khomh. Characterizing faults in agentic ai: A taxonomy of types, symptoms, and root causes. *arXiv preprint arXiv:2603.06847*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968, 2024.

Lingzhe Zhang, Tong Jia, Mingyu Wang, Weijie Hong, Chiming Duan, Minghua He, Rongqian Wang, Xi Peng, Meiling Wang, Nicholas Zhang, et al. Efficient failure management for multi-agent systems with reasoning trace representation. In *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering*, pages 1222–1226, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which

agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 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*, 2026a.

Chenyu Zhao, Shenglin Zhang, Yihang Lin, Wenwei Gu, Zhimin Chen, Yongqian Sun, Dan Pei, Chetan Bansal, Saravan Rajmohan, and Minghua Ma. Debugging the debuggers: Failure-anchored structured recovery for software engineering agents. *arXiv preprint arXiv:2605.08717*, 2026b.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606, 2024.

A Replay Matching Rules and Runtime Assumptions

A.1 Observed Execution Order

The event-dependency graph $G = (V, E)$ defines a partial order but does not uniquely determine a concurrent execution. Consistent with the main paper, SymTrace therefore records the observed event order

$$O = (v_1, \ldots, v_n), \tag{4}$$

subject to

$$(v_i, v_j) \in E \Longrightarrow i < j. \tag{5}$$

Unlike a topological ordering computed after execution, O is the particular linear extension observed during the recorded run. It fixes the order in which state-affecting operations crossed an instrumented boundary. During replay, the recorded order is used solely for event matching and validation; the adapter does not replace or control the native MAS scheduler.

Algorithm 1 records exposed events and boundary results without replacing native MAS control flow.

A.2 Boundary Matching and Record Validation

The ordered boundary records R in the replay bundle preserve complete interactions. An LLM entry contains its prompt and ordered messages, model configuration, tool specification, expected response format, and returned response. A tool entry contains the tool identity, arguments, returned observation, and execution metadata. Each entry also retains its position in O.

For an intercepted boundary event v_j, the adapter validates the request against the expected record using its position in O and canonicalized request content. A mismatch terminates replay at the first divergence, whereas a successful match authorizes the recorded result to be returned through the native boundary. After the result is materialized, its node content hash is compared with the source record to validate the reused prefix.

Algorithm 2 gives the fail-closed procedure for selective replay. A mismatch terminates at the first divergence rather than scanning forward for another record.

In selective mode, Algorithm 2 stops before a designated LLM event v_k and augments its boundary request with repair guidance Δ,

$$\widehat{q}_k = q_k \oplus \Delta, \tag{6}$$

Algorithm 1 Replay Bundle Recording

Require: Task T, represented initial state s_0, runtime configuration c
Ensure: Replay bundle $\mathcal{S} = \langle T, s_0, c, G, O, R \rangle$
Initialize $V, E \leftarrow \emptyset$ and $O, R \leftarrow ()$
Start the native MAS with the framework adapter enabled
while the native MAS has not terminated **do**
  Observe event v and assign its identity, agent, type, and ordinal
  Add v and its dependencies to $G = (V, E)$; append v to O
  if v crosses an LLM or tool boundary **then**
    Capture request x_v and configuration λ_v
    Execute the live boundary to obtain y_v
    Append (v, x_v, λ_v, y_v) to R and return y_v
  end if
end while
return $\mathcal{S} = \langle T, s_0, c, G, O, R \rangle$

before switching the target and suffix to live execution. The represented history before v_k is preserved, while intentional divergence begins at the intervention boundary.

A.3 Replay Conditions and Scope

The replay guarantee relies on four conditions:

1. the task, initialization, executable code, and runtime configuration are fixed;
2. internal MAS transitions are deterministic when conditioned on the same represented state, input, and boundary result;
3. every state-affecting model response, tool observation, external input, and scheduling decision is captured; and
4. replay returns a recorded result only after validating the event position and canonicalized request content, and separately validates each materialized reused prefix node using its content hash.

These conditions do not require the live model or external service itself to be deterministic. They make explicit the replay guarantee summarized in the main paper: selective replay reconstructs the represented logical execution prefix recorded in the replay bundle.

Although arbitrary hidden framework objects are not serialized, their represented logical values are reconstructed by native execution because they undergo the same transitions from the same initial state. The guarantee covers recorded boundary results, agent-visible messages, represented logical state, control decisions, and event order. It does not require

Algorithm 2 Fail-Closed Selective Replay

Require: Replay bundle $\mathcal{S} = \langle T, s_0, c, G, O, R \rangle$
Require: Optional target v_k and repair guidance Δ
Ensure: Replayed trace or first-divergence report

Restart the native MAS from (T, s_0, c)
$j \leftarrow 1$; *live* $\leftarrow$ false
while the MAS requests a model or tool boundary v **do**
  if *live* **then**
    Execute v live and return its result
  else if $v = v_k$ **then**
    Augment the request: $\widehat{q}_k \leftarrow q_k \oplus \Delta$
    live $\leftarrow$ true
    Execute the target boundary live using $\widehat{q}_k$
  else
    if $j > |R|$ or the event position or canonicalized request content does not match R_j **then**
      terminate and report the first divergence
    end if
    Return the recorded result in R_j through the native boundary
    Validate the materialized event position and content hash
    $j \leftarrow j + 1$
  end if
end while
return the materialized execution trace

identical memory addresses, physical thread interleavings, network latency, or wall-clock timing.

A.4 External State Considerations

A recorded tool result reconstructs the observation available to the MAS but does not reproduce a persistent side effect in an external service. Replaying a stored success response for a write operation, for example, does not recreate the written object in a live backend. Selective repair therefore requires that the live suffix not depend on an unreproduced mutation in the replayed prefix. This condition holds when the prefix contains no required persistent write, the environment is reset and the write can be safely re-executed, or a service-level checkpoint restores the required state. The reported evaluation includes all 536 source failures without filtering cases according to external-state dependence; when relevant external state is not reset, isolated, or restored, any resulting suffix variation lies outside the replay guarantee.

B Dataset Construction, Annotation, and Reliability

B.1 Cohort Accounting

Applying the three MASs to the 200 fixed tasks produces 600 initial task–MAS executions. The native benchmark evaluators identify 536 failed executions with complete trace and graph artifacts, all of which enter SymFail and are subsequently annotated. The remaining 64 executions are evaluator-accepted and are excluded before annotation. The finalized dataset therefore contains 536 failures: 462 from WebArena-Verified and 74 from AssistantBench, comprising 171 AG2, 184 CrewAI, and 181 Magentic-One executions.

B.2 Annotation Rules

Multi-label taxonomy. The C1–C4 categories follow the diagnosis-to-repair mapping in the main paper. They are non-exclusive, trace-observable operational symptoms rather than mutually exclusive causal mechanisms, so annotators assign every category directly supported by the localized trace and separately select one primary category. The primary category is the best-supported intervention signal at the localized node; the guide imposes no category priority and does not treat it as a unique root cause. Dataset-level failure composition is computed from the complete category sets.

The four symptom categories are derived from the following source mappings:

- **C1 (Task-Constraint Violation)** corresponds to category 1.1 Disobey Task Specification from the initial taxonomy.
- **C2 (Repeated or Stalled Progress)** merges categories 1.3 Step Repetition and 1.5 Unaware of Termination Conditions.
- **C3 (Unresolved Runtime Condition)** originates from categories 2.2 Fail to Ask for Clarification and 2.3 Task Derailment, and is extended in this work to cover execution or context failures.
- **C4 (Plan–Action–Outcome Inconsistency)** corresponds to category 2.6 Action–Reasoning Mismatch.

The boundary between C2 and C3 receives an additional decision rule because repetition is especially interpretation-sensitive. C2 requires repeated or semantically equivalent behavior after the trace has established that the current plan cannot succeed. A first failed action, one justified retry, a retry based

on new parameters or evidence, or a substantive refinement is insufficient. An unresolved dependency without repetition is assigned C3 rather than C2.

Failure-node localization. Annotators choose the earliest trace-supported node at which an error is both actionable and connected through the recorded execution to the evaluator-confirmed failure. An `agent_action` node is used when an agent decision commits the error; a `tool_call` node when an erroneous or unjustified invocation commits it; a `tool_result` node when a returned result or execution failure first becomes a decisive unhandled state; and a `final_result` node when the error is introduced, or becomes decidable, only in the final response. Later repetitions or propagations of the same failure do not replace the earliest actionable node. Multiple symptoms at that node are encoded as multiple categories, not multiple primary nodes.

Annotation records and confidence. For each case, an annotator records the complete category set, primary category, failure-node ID and type, confidence, rationale, and supporting trace evidence. Confidence describes evidential clarity rather than a probability or voting weight: *high* denotes direct support without a comparably supported alternative, *medium* denotes a plausible alternative interpretation, and *low* denotes insufficient evidence for a defensible tuple without further inspection. Confidence is retained as review metadata and does not mechanically affect adjudication.

B.3 Annotation Procedure

Four annotators with backgrounds in software engineering participated in the annotation process. Before annotating the full dataset, each annotator completed a calibration procedure consisting of reference-labeled examples and practice trajectories to establish consistent interpretation of the category definitions and localization rules.

Three annotators independently annotated each case. The fourth annotator then examined the complete traces, all three independent annotations, and the supporting evidence records, and produced the final annotation for each case. The final annotation consists of the strict tuple

$$(\text{exact category set},\ \text{primary category},\ \text{node ID},\ \text{node type}).$$

Table 5 summarizes the relationship between the initial independent annotations and the final annotation.

The final annotation is supported in full by three initial annotators in 187 cases (34.89%), two in 124 (23.13%), one in 125 (23.32%), and none in 100 (18.66%). The final category set equals the three-annotator union in 423 cases (78.92%) and differs in 113 (21.08%). Across the latter decisions, the fourth annotator adds 123 case–category assignments absent from all initial annotations and removes 69 assignments proposed by at least one annotator, including 16 C1 assignments initially supported by all three. These bidirectional changes show that the final annotation re-evaluates trace evidence rather than merely aggregating votes.

Table 5: Relation between the three independent annotations and the final annotation. "Proposed" denotes a complete strict tuple submitted by at least one annotator.

Initial pattern	Cases	Outcome relative to final annotation
Unanimous	210	all three retained 187; modified 23
2–1 split	214	majority retained 124; minority retained 48; new tuple 42
All distinct	112	one proposed tuple retained 77; new tuple 35

Table 6: Agreement among the three independent human annotators. Pairwise agreement averages the three annotator pairs; 3/3 agreement is the number of cases in which all annotators make the same decision.

Annotation target	Fleiss' κ (95% CI)	Pairwise agreement	3/3 agreement
Primary category	0.622 (0.545–0.692)	89.86%	459/536 (85.63%)
Exact multi-label category set	0.574 (0.535–0.612)	64.74%	280/536 (52.24%)
C1 presence	0.696 (0.617–0.769)	93.91%	487/536 (90.86%)
C2 presence	0.374 (0.306–0.440)	79.23%	369/536 (68.84%)
C3 presence	0.718 (0.671–0.764)	86.19%	425/536 (79.29%)
C4 presence	0.688 (0.628–0.745)	88.43%	443/536 (82.65%)
Failure-node type	0.811 (0.759–0.857)	93.97%	488/536 (91.04%)

B.4 Inter-Annotator Reliability

Reliability is computed from the three initial annotations for all 536 cases. Primary categories and node types are nominal variables. For the non-exclusive taxonomy, we calculate Fleiss' κ both for the exact category combination as a nominal outcome and for the presence of each C1–C4 category as a binary decision. Percentile 95% confidence intervals use 20,000 case-level bootstrap resamples with seed `20260722`; resampling cases preserves the three associated ratings. Final annotations are excluded from reliability calculations.

Failure-node IDs are case-specific and do not share a nominal category space, so Fleiss' κ is not defined for this target. Exact agreement is reported instead: all three annotators select the same node in 396/536 cases (73.88%; Wilson 95% CI: 70.00%–77.42%), and at least two select the same node in 514/536 (95.90%). For the complete strict tuple, all three annotations agree in 210/536 cases (39.18%; Wilson 95% CI: 35.14%–43.37%), and at least two agree in 424/536

(79.10%).

C2 has the lowest category-specific reliability ($\kappa = 0.374$) and accounts for 86 of the 123 added category decisions in the final annotation. This concentration identifies the repeated-or-stalled-progress boundary as the principal taxonomy-level ambiguity and motivates the explicit C2 decision rule above.

B.5 Final Composition and Integrity Checks

The 536 finalized cases contain 1,482 category assignments, with mean category cardinality 2.77. The primary-category distribution is C1 = 125 (23.32%), C2 = 69 (12.87%), C3 = 203 (37.87%), and C4 = 139 (25.93%). Multi-label marginal counts are C1 = 439, C2 = 288, C3 = 289, and C4 = 466; because categories are non-exclusive, these counts exceed the number of cases.

Every selected node is validated against its corresponding event graph. Validation requires the graph to exist, the node ID to occur in the graph, and the recorded node type to match the graph node. All 1,608 initial selections and 536 final selections satisfy these checks, for 2,144/2,144 validated selections in total.

SymFail is scoped to two benchmark sources, three MASs, and 536 evaluator-confirmed failures with complete trace and graph artifacts. The finalized fields are human-adjudicated references, not outputs from the LLM annotation procedure evaluated later. We release the linked traces and graphs, all initial annotations and evidence records, and the adjudicated fields to support independent inspection; broader independent re-annotation remains future work.

C Execution and Statistical Conventions

C.1 Repair Conditions and Execution Limits

Paired methods are run on matched source failures. The task input, MAS implementation, recorded model alias, temperature, and native evaluator are held fixed within a pair. Conditions differ only in the information supplied by the repair prompt, the intervention location, and whether and where the recorded trajectory is replayed. Multiple internal model calls, tool calls, or technical retries within one top-level invocation are not treated as independent observations.

For RQ1, each source failure is evaluated through three executions per method. For RQ2, each task-level method receives up to three complete attempts and stops after its first evaluator-accepted result. For RQ3, each node-level method receives one selective-replay intervention, whereas the task-level baselines retain their up-to-three-attempt budget. A case is considered repaired if at least one permitted attempt passes the native evaluator. Actual API-call counts may differ because the methods generate different trajectories; efficiency is therefore measured rather than inferred from the nominal attempt budgets.

C.2 Framework and Model Records

The study uses AG2 0.13.3. The vendored lock installs the CrewAI main package as an editable project and therefore does not attach a version field to that lock entry; the main package's own `crewai/__init__.py` reports version 1.14.7a3, and its project metadata pins `crewai-core` and `crewai-cli` to the same version. The separately named `crewai-tools` package in the vendored workspace also reports version 1.14.7a3. The MagenticOne adapter uses AutoGen AgentChat 0.7.5. Its execution path imports `AssistantAgent` and `MagenticOneGroupChat` from `autogen_agentchat`. The model client, `OpenAIChatCompletionClient`, comes from `autogen_ext`. The adapter does not import `pyautogen`; the vendored lock records `pyautogen` 0.10.0 only as a meta-package whose dependency is `autogen-agentchat` 0.7.5.

The repair experiments were conducted from June 15 to June 16, 2026 UTC. Per-MAS execution windows are retained in the artifact.

All requests use temperature 0.00 through an OpenAI-compatible hosted endpoint. The requested model alias is `deepseek-v4-flash`. These values describe the recorded client request, not a verified immutable model snapshot. The experiment records do not contain a confirmed upstream provider identity or server-side revision. We therefore avoid attributing reproducibility to the alias or temperature setting alone.

C.3 Statistical Procedures

For a repair proportion $\hat{p}$, we report a 95% Wilson confidence interval. For two methods evaluated on matched source failures, the effect size is the absolute repair-rate difference

$$\Delta = 100(\hat{p}_{\mathrm{s}} - \hat{p}_{\mathrm{b}})$$

in percentage points. Its 95% confidence interval is estimated from 10,000 case-level paired bootstrap resamples using seed `20260701`. The two method outcomes for a case are resampled together to preserve pairing.

RQ1 compares SymTrace Replay with Unguided Full Rerun using a two-sided exact McNemar test on the paired per-case rep_3 outcomes. RQ2 compares Self-Reflection and Critic-Agent separately with Unguided Full Rerun within each MAS; the two raw p-values form one prespecified Holm family per MAS. RQ3 compares Suspicious-Node Intervention with Unguided Full Rerun, Self-Reflection, Critic-Agent, Last-Node Intervention, and Random-Node Intervention within each MAS; the five raw p-values form one prespecified Holm family per MAS. Since there are three MASs, RQ3 contains three separate five-comparison Holm families totalling 15 tests. Holm-adjusted values are denoted by p_{H}.

For every repair-method comparison, we report the matched denominator, both repair counts and rates, Δ and its paired confidence interval, and the applicable adjusted value. Each source failure contributes at most one binary outcome per method to a comparison; internal steps, calls, and technical retries are not analyzed as independent samples.

C.4 Paired Comparison Results

All confidence intervals reported below are percentile intervals obtained from 10,000 case-level paired bootstrap resamples. McNemar tests are exact and two-sided. System and framework errors remain in the denominator and are counted as unsuccessful repairs. For RQ2, Holm correction is applied separately within each MAS to the two comparisons of Self-Reflection and Critic-Agent against Unguided Full Rerun. For RQ3, Holm correction is applied separately within each MAS to the five comparisons involving Suspicious-Node Intervention.

Across all six RQ2 comparisons, the repair-rate differences relative to Unguided Full Rerun are negative, and none is statistically significant after within-MAS Holm correction. Thus, the results provide no evidence that Self-Reflection or Critic-Agent outperforms unguided rerunning under the same three-attempt budget.

Across all 15 RQ3 comparisons, Suspicious-Node Intervention achieves a positive repair-rate difference whose 95% confidence interval excludes zero. All comparisons remain statistically significant after within-MAS Holm correction ($p_{\text{H}} < 0.05$). Thus, the advantage of Suspicious-Node Intervention is consistent across the three MAS architectures and against both task-level and node-level controls.

C.5 API-Call Efficiency

Table 9 reports API-call efficiency as a secondary descriptive measure. A repair is counted only when `recovered=true`, indicating that the generated execution is accepted by the native evaluator. Cases ending in system or framework errors are counted as non-repairs, and any API calls made before those errors remain in the API-call total. For method m, we compute

$$\text{RepairsPer1K}(m) = \frac{\text{Repairs}(m)}{\text{APICalls}(m)} \times 1000, \quad (7)$$

and

$$\text{CallsPerRepair}(m) = \frac{\text{APICalls}(m)}{\text{Repairs}(m)}. \quad (8)$$

Because methods may generate trajectories of different lengths and receive different attempt budgets, these measures are compared primarily within each MAS and are not treated as compute-matched causal estimates.

Across AG2, CrewAI, and Magentic-One, Suspicious-Node achieves both the highest number of accepted repairs per 1,000 API calls and the lowest number of API calls per repair. These results provide consistent secondary evidence that symptom-guided localized intervention yields greater repair efficiency than the task-level and node-level controls within each MAS.

C.6 Implementation Environment

The experiments are orchestrated with Python 3.11.15 on Windows 11 using an Intel Core i9-14900HX processor, 32 GB of RAM, and an NVIDIA GeForce RTX 4060 Laptop GPU with 8 GB of VRAM. Model inference is performed by the hosted endpoint. The local workstation runs the MAS frameworks and tools, trace capture, SymTrace replay, native evaluation, and statistical analysis.

D Reproduction Settings and Fidelity

D.1 Mechanism-Level Failure Equivalence

The formal equivalence test used by the method-blinded judges is retained below. For attempt a of

Table 7: Paired RQ2 comparisons against Unguided Full Rerun. Δ is the repair-rate difference between the evaluated method and Unguided Full Rerun. n_{10}/n_{01} denotes Method-only/Rerun-only repairs. Each Holm family contains the Self-Reflection and Critic-Agent comparisons within one MAS.

MAS	Method	Method pass@3	Rerun pass@3	Δ pp [95% CI]	n_{10}/n_{01}	Raw p	p_{H}
AG2	Self-Reflection	8/171 (4.68%)	14/171 (8.19%)	−3.51 [−8.19, 1.17]	5/11	0.210	0.237
AG2	Critic-Agent	7/171 (4.09%)	14/171 (8.19%)	−4.09 [−8.77, 0.00]	4/11	0.118	0.237
Magentic-One	Self-Reflection	11/181 (6.08%)	16/181 (8.84%)	−2.76 [−7.73, 2.21]	8/13	0.383	0.383
Magentic-One	Critic-Agent	8/181 (4.42%)	16/181 (8.84%)	−4.42 [−9.39, 0.00]	6/14	0.115	0.231
CrewAI	Self-Reflection	4/184 (2.17%)	7/184 (3.80%)	−1.63 [−4.35, 1.09]	2/5	0.453	0.906
CrewAI	Critic-Agent	5/184 (2.72%)	7/184 (3.80%)	−1.09 [−4.35, 1.63]	3/5	0.727	0.906

Table 8: Paired RQ3 comparisons. Δ is the repair-rate difference between Suspicious-Node Intervention and the comparator. n_{10}/n_{01} denotes Suspicious-only/Comparator-only repairs. Suspicious-Node, Random-Node, and Last-Node use pass@1, whereas the three task-level comparators use pass@3, consistent with the budgets in the main paper. Each Holm family contains the five comparisons within one MAS.

MAS	Comparator	Suspicious-Node	Comparator	Δ pp [95% CI]	n_{10}/n_{01}	Raw p	p_{H}
AG2	Unguided Full Rerun	28/171 (16.37%)	14/171 (8.19%)	+8.19 [1.75, 14.62]	24/10	0.0243	0.0243
AG2	Self-Reflection	28/171 (16.37%)	8/171 (4.68%)	+11.70 [5.26, 18.13]	26/6	5.35×10^{-4}	1.07×10^{-3}
AG2	Critic-Agent	28/171 (16.37%)	7/171 (4.09%)	+12.28 [7.02, 18.13]	24/3	4.92×10^{-5}	1.48×10^{-4}
AG2	Random-Node	28/171 (16.37%)	4/171 (2.34%)	+14.04 [8.19, 19.88]	27/3	8.43×10^{-6}	3.37×10^{-5}
AG2	Last-Node	28/171 (16.37%)	1/171 (0.58%)	+15.79 [10.53, 21.64]	27/0	1.49×10^{-8}	7.45×10^{-8}
Magentic-One	Unguided Full Rerun	34/181 (18.78%)	16/181 (8.84%)	+9.94 [3.87, 16.02]	27/9	0.00393	0.00393
Magentic-One	Self-Reflection	34/181 (18.78%)	11/181 (6.08%)	+12.71 [6.63, 18.78]	29/6	1.17×10^{-4}	2.34×10^{-4}
Magentic-One	Critic-Agent	34/181 (18.78%)	8/181 (4.42%)	+14.36 [8.29, 20.44]	30/4	6.16×10^{-6}	1.85×10^{-5}
Magentic-One	Random-Node	34/181 (18.78%)	7/181 (3.87%)	+14.92 [9.39, 20.44]	29/2	4.63×10^{-7}	1.85×10^{-6}
Magentic-One	Last-Node	34/181 (18.78%)	3/181 (1.66%)	+17.13 [11.05, 23.20]	33/2	3.67×10^{-8}	1.84×10^{-7}
CrewAI	Unguided Full Rerun	46/184 (25.00%)	7/184 (3.80%)	+21.20 [14.67, 27.72]	43/4	2.78×10^{-9}	5.56×10^{-9}
CrewAI	Self-Reflection	46/184 (25.00%)	4/184 (2.17%)	+22.83 [16.85, 29.35]	43/1	5.12×10^{-12}	2.05×10^{-11}
CrewAI	Critic-Agent	46/184 (25.00%)	5/184 (2.72%)	+22.28 [15.76, 28.80]	43/2	5.89×10^{-11}	1.77×10^{-10}
CrewAI	Random-Node	46/184 (25.00%)	9/184 (4.89%)	+20.11 [13.59, 26.63]	40/3	3.02×10^{-9}	5.56×10^{-9}
CrewAI	Last-Node	46/184 (25.00%)	3/184 (1.63%)	+23.37 [17.39, 29.89]	44/1	2.61×10^{-12}	1.31×10^{-11}

method m, the category-matching events are

$$\mathcal{M}^{c}_{i,a,m} = \left\{ \widehat{e} \in \widehat{\tau}^{m}_{i,a} : \widehat{c}(\widehat{e}) = c_i \right\}. \tag{9}$$

The mechanism-matching and role-matching sets are

$$\mathcal{M}^{m}_{i,a,m} = \left\{ \widehat{e} \in \widehat{\tau}^{m}_{i,a} : \widehat{m}(\widehat{e}) \equiv m_i \right\}, \tag{10}$$

$$\mathcal{M}^{\rho}_{i,a,m} = \left\{ \widehat{e} \in \widehat{\tau}^{m}_{i,a} : \widehat{\rho}(\widehat{e}) \sim \rho_i \right\}. \tag{11}$$

Their intersection contains events satisfying all three semantic criteria:

$$\mathcal{M}_{i,a,m} = \mathcal{M}^{c}_{i,a,m} \cap \mathcal{M}^{m}_{i,a,m} \cap \mathcal{M}^{\rho}_{i,a,m}. \tag{12}$$

The attempt reproduces the source mechanism only when at least one such event also has grounded evidence:

$$z^{m}_{i,a} = \mathbf{1}\left[\exists \widehat{e} \in \mathcal{M}_{i,a,m} : \mathrm{Grounded}(\widehat{E}_{\widehat{e}}, E_i) \right]. \tag{13}$$

Here, c_i, m_i, ρ_i, and E_i denote the source category, case-specific mechanism, semantic execution role, and supporting evidence. The relations $\equiv$ and $\sim$ express mechanism equivalence and semantic-role correspondence rather than raw node equality.

D.2 Reproduction Metrics

Let N be the number of source cases. Per-execution reproduction is

$$\mathrm{rep}_1(m) = \frac{1}{3N} \sum_{i=1}^{N} \sum_{a=1}^{3} z^{m}_{i,a}. \tag{14}$$

Reproduction in all three attempts is

$$\mathrm{rep}_3(m) = \frac{1}{N} \sum_{i=1}^{N} \prod_{a=1}^{3} z^{m}_{i,a}. \tag{15}$$

D.3 Replay-Fidelity Audit

Failure recurrence does not itself establish that replay preserved the historical context. Before the target, every model or tool request is matched against the ordered replay bundle using its event position and canonicalized request content. A missing, out-of-order, or non-matching request terminates replay rather than silently continuing.

We separately compare the materialized replay graph with the source prefix. Across 536 cases

Table 9: API-call efficiency on the three MASs evaluated in the main paper. Repairs count native-evaluator-accepted executions. Higher Repairs/1K calls and lower Calls/repair indicate greater repair yield per API call. Bold marks the best value within each MAS.

MAS	Method	API calls	Repairs	Repairs/1K calls	Calls/repair
AG2	Last-Node	249	1	4.0	249.0
AG2	Random-Node	293	4	13.7	73.3
AG2	Critic-Agent	295	7	23.7	42.1
AG2	Self-Reflection	314	8	25.5	39.3
AG2	Unguided Full Rerun	400	14	35.0	28.6
AG2	Suspicious-Node	419	28	**66.8**	**15.0**
CrewAI	Last-Node	227	3	13.2	75.7
CrewAI	Random-Node	251	9	35.9	27.9
CrewAI	Critic-Agent	256	5	19.5	51.2
CrewAI	Self-Reflection	258	4	15.5	64.5
CrewAI	Unguided Full Rerun	254	7	27.6	36.3
CrewAI	Suspicious-Node	466	46	**98.7**	**10.1**
Magentic-One	Last-Node	239	3	12.6	79.7
Magentic-One	Random-Node	278	7	25.2	39.7
Magentic-One	Critic-Agent	324	8	24.7	40.5
Magentic-One	Self-Reflection	343	11	32.1	31.2
Magentic-One	Unguided Full Rerun	367	16	43.6	22.9
Magentic-One	Suspicious-Node	443	34	**76.7**	**13.0**

and three attempts per case, every replay plan uses the exact target ID from the corresponding human-adjudicated source annotation (1,608/1,608). Every attempt also preserves the parent/edge topology of the reused prefix, and all 6,159 materialized reused nodes match their source content hashes (6,159/6,159). Attempt-level prefix exactness is therefore 1,608/1,608. Because the target is supplied from the human-adjudicated annotation, this audit measures replay fidelity rather than automatic localization accuracy.

These audits rule out modified recorded-prefix content as the source of reproduction below 100.00%. Remaining variation may arise at the regenerated target, in subsequent live execution, or from external state outside the replay boundary.

E Task-Level Repair Aggregation

The main paper defines the task-level strategies, evaluation sets, budgets, and results. We retain only the formal pass@3 aggregation. Let $\mathcal{F}$ denote an evaluation set of initially failed executions, $y_{i,a}^m$ the evaluator outcome of attempt a, and $A_i^m \leq 3$ the attempts executed before success or exhaustion. The terminal case outcome is

$$R_i^m = \mathbf{1}\left[\max_{1\leq a\leq A_i^m} y_{i,a}^m = 1\right]. \tag{16}$$

Cumulative repair is

$$\text{pass@3}(m) = \frac{1}{|\mathcal{F}|}\sum_{i\in\mathcal{F}} R_i^m. \tag{17}$$

E.1 Baseline Input Boundaries and Prompt Templates

The three task-level baselines are evaluated on the same 536 source failures from AG2, CrewAI, and Magentic-One. All three receive the task specification, task identifier, benchmark and split identifiers, and the available start URLs or tools. Self-Reflection and Critic-Agent additionally receive the final answer from the original failed execution. None of the baselines receives the source trajectory or event graph, failure location, C1–C4 category, trace evidence, reference or gold answer, or evaluator information.

For every recovery attempt, Self-Reflection and Critic-Agent reuse the original failed execution's final answer rather than the output of a preceding

recovery attempt. Critic-Agent implements the dedicated critic role described in the main paper through a fixed critic-role prompt within each complete rerun attempt; it does not introduce an additional standalone critic-model call. Apart from their method-specific feedback, all three baselines use the same task information, output suffix, attempt budget, model configuration, and native evaluator.

Unguided Full Rerun.

```
Mode: fail-then-rerun
Task id: {task_id}
Benchmark: {source_benchmark} / {source_split}
Task: {task}
Start URLs or tools: {start_urls}
This is an unguided rerun after an initial failed or
uncertain attempt.
Solve from scratch.
{output_suffix}
```

Self-Reflection.

```
Mode: self-reflection
Task id: {task_id}
Benchmark: {source_benchmark} / {source_split}
Task: {task}
Start URLs or tools: {start_urls}
Previous final answer:
{previous_final_answer}
Reflect on why the previous attempt may be incomplete
or wrong, then solve the task again.
{output_suffix}
```

Critic-Agent.

```
Mode: critic-agent
Task id: {task_id}
Benchmark: {source_benchmark} / {source_split}
Task: {task}
Start URLs or tools: {start_urls}
Previous final answer:
{previous_final_answer}
Critic feedback: verify assumptions, check whether the
cited evidence is sufficient, and correct any
     unsupported
or stale claims before producing the new answer.
{output_suffix}
```

For the WebArena-Verified and AssistantBench tasks used in this study, `{output_suffix}` is instantiated as follows:

```
Return only the final answer and a short note about what
evidence would be needed.
```

Because the three task-level baselines share the same task inputs, output suffix, model configuration, evaluator, and three-attempt budget, their comparison isolates whether access to the original failed answer and generic reflection or critic guidance provides an advantage over unguided rerunning. The paired results in Appendix C.4 show no such advantage for Self-Reflection or Critic-Agent.

F Detection Rules, Anchor Ranking, and Intervention Prompts

F.1 Rule-First Symptom Detection

Let $G = (V, E)$ be the event-dependency graph, T the task specification, and $\tau_{<v}$ the trace preceding node v. For category C_k, define its trigger indicator as

$$b_k(v) = \bigvee_{r \in \mathcal{R}_k} r(v, \tau_{<v}, T). \tag{18}$$

The deterministic candidate set collects the triggered categories:

$$L_v^{\mathrm{r}} = \{C_k : b_k(v) = 1\}. \tag{19}$$

Algorithm 3 applies these conditions to every completed node. Nodes without a deterministic trigger receive zero suspiciousness without invoking the semantic judge. For triggered nodes, the judge may confirm or reject supplied categories but must satisfy

$$L_v^{\mathrm{f}} \subseteq L_v^{\mathrm{r}}. \tag{20}$$

We write $L_v = L_v^{\mathrm{f}}$ for the confirmed category set used by the ranking procedure. Reference answers, evaluator verdicts, and evaluator-derived evidence are removed before verification, so both rule evidence and semantic judgments use only the task and runtime trace.

C1–C4 are non-exclusive operational symptoms. A category can denote a local error, unresolved dependency, or downstream manifestation, so it raises intervention priority without asserting that the node is the unique root cause.

F.2 Repairability-Aware Anchor Selection

A strongly suspicious node need not be the best intervention anchor: terminal nodes leave little suffix to repair, while propagation-only nodes may merely expose earlier failures. Let $c_{\mathrm{l}}(v)$ be the strongest locally supported confidence, $c_{\mathrm{r}}(v)$ include propagated evidence, $I_{\mathrm{p}}(v)$ indicate propagation-only evidence, and $\rho(v) \in [0, 1]$ be normalized graph position.

The repairability score begins with node type and local evidence:

$$B_{\mathrm{r}}(v) = \beta_0 + \eta(v) + \beta_{\mathrm{l}} c_{\mathrm{l}}(v). \tag{21}$$

Symptom multiplicity contributes

$$G_{\mathrm{multi}}(v) = \beta_{\mathrm{m}} \mathbf{1}[|L_v| > 1]. \tag{22}$$

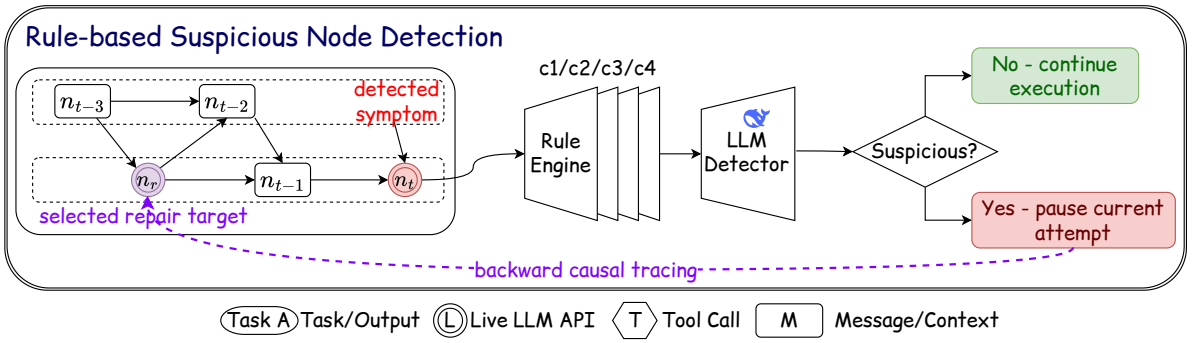


Figure 3: Suspicious-Node Intervention pipeline. Deterministic rules and a semantic judge jointly evaluate each newly completed node for the C1–C4 symptoms. When the fused suspicion score exceeds the predefined threshold, the ongoing MAS execution is suspended. SymTrace then reconstructs the execution prefix preceding the selected intervention anchor, injects symptom-conditioned repair guidance derived from the triggered rules, and resumes live execution to generate a new downstream trajectory.

Table 10: Deterministic conditions used to generate trace-localized symptom candidates. The semantic judge may reject a candidate but cannot introduce a category outside this set.

Category	Operational meaning	Deterministic candidate conditions
C1: Task-constraint violation	The behavior conflicts with an explicit task constraint or output requirement.	The node violates a stated constraint, produces an incompatible output format, contains a placeholder or skipped action, or returns an unresolved request instead of the required result.
C2: Repeated or stalled progress	The history repeats while the objective remains unfinished.	The normalized action fingerprint or tool URL matches a preceding node; the node retries without new parameters or evidence; or another final-style output follows without an action that advances the unfinished objective.
C3: Unresolved runtime condition	The node contains or continues from an unresolved execution, access, or upstream condition.	The node records an error, timeout, or failed tool result; reports inaccessible required evidence; or proceeds toward a final answer while an error remains in the preceding trace.
C4: Plan–action–outcome inconsistency	An explicit plan or claim conflicts with the executed action or observed outcome.	A browser action lacks a URL or uses an invalid `view-source:` URL; structured data is requested but HTML is returned; completion follows an unsuccessful action; or a tool-dependent task is answered without successful tool evidence.

The C2 and C4 category boost is

$$G_{\text{cat}}(v) = \beta_{C_4}\mathbf{1}[C_4 \in L_v] + \beta_{C_2}\mathbf{1}[C_2 \in L_v]. \quad (23)$$

Their sum forms the positive repairability adjustment:

$$G_{\text{r}}(v) = G_{\text{multi}}(v) + G_{\text{cat}}(v). \quad (24)$$

Propagation-only evidence and late graph position contribute the penalty

$$P_{\text{r}}(v) = \beta_{\text{p}} I_{\text{p}}(v) + \beta_{\pi}\rho(v). \quad (25)$$

The complete repairability score is

$$S_{\text{r}}(v) = \text{clip}_{[0,1]}\left(B_{\text{r}}(v) + G_{\text{r}}(v) - P_{\text{r}}(v)\right). \quad (26)$$

Here, $\eta(v)$ is a node-type prior. Regenerable agent actions receive the highest prior, followed by tool calls and results, while final-result nodes receive the lowest.

Among replayable nodes, the selection score first combines repairability with local and propagated confidence:

$$B_{\text{s}}(v) = \alpha_{\text{r}} S_{\text{r}}(v) + \alpha_{\text{l}} c_{\text{l}}(v) + \alpha_{\text{u}} c_{\text{r}}(v). \quad (27)$$

The positive selection adjustment is

$$G_{\text{s}}(v) = \alpha_{\text{n}} \min(2, |L_v|) + \alpha_{\text{a}} I_{\text{a}}(v), \quad (28)$$

and the selection penalty is

$$P_{\text{s}}(v) = \alpha_{\text{p}} I_{\text{p}}(v) + \alpha_{\text{f}} I_{\text{f}}(v) + \alpha_{\pi}\rho(v). \quad (29)$$

The complete selection score is

$$S_{\text{s}}(v) = \text{clip}_{[0,1]}\left(B_{\text{s}}(v) + G_{\text{s}}(v) - P_{\text{s}}(v)\right). \quad (30)$$

Here, $I_{\text{a}}(v)$ indicates an actionable `agent_action` or `tool_call` boundary and $I_{\text{f}}(v)$ indicates a

Algorithm 3 Rule-Gated Symptom Detection

Require: Task T, event-dependency graph G, observed order O
Require: Rule sets $\{\mathcal{R}_1, \ldots, \mathcal{R}_4\}$ and semantic judge $\mathcal{J}$
Ensure: Confirmed candidate set $\mathcal{C}$

$\mathcal{C} \leftarrow \emptyset$
for each completed node v in O **do**
  Construct the diagnostic view from $(T, v, \tau_{<v}, G)$
  $L_v^{\mathrm{r}} \leftarrow \emptyset$; $E_v^{\mathrm{r}} \leftarrow \emptyset$
  for $k = 1$ to 4 **do**
    for each rule $r \in \mathcal{R}_k$ **do**
      if $r(v, \tau_{<v}, T) = 1$ **then**
        Add C_k and its matched evidence to $(L_v^{\mathrm{r}}, E_v^{\mathrm{r}})$
      end if
    end for
  end for
  if $L_v^{\mathrm{r}} \neq \emptyset$ **then**
    Remove all reference-answer and evaluator information
    Query $\mathcal{J}$ for $(L_v^{\mathrm{f}}, \widetilde{c}_{\mathrm{LLM}}(v), \xi_v)$
    $L_v^{\mathrm{f}} \leftarrow L_v^{\mathrm{f}} \cap L_v^{\mathrm{r}}$
    if $L_v^{\mathrm{f}} \neq \emptyset$ **then**
      $E_v^{\mathrm{f}} \leftarrow$ evidence in E_v^{r} supporting L_v^{f}
      Add $(v, L_v^{\mathrm{f}}, E_v^{\mathrm{f}}, \widetilde{c}_{\mathrm{LLM}}(v))$ to $\mathcal{C}$
      Retain ξ_v for audit only
    end if
  end if
end for
return $\mathcal{C}$

`final_result` node. The coefficient design prioritizes repairability, followed by local evidence and then propagated evidence.

Semantic confidence and the structured score are first mixed as

$$c_{\mathrm{mix}}(v) = 0.55\, \widetilde{c}_{\mathrm{LLM}}(v) + 0.45\, S_{\mathrm{s}}(v). \tag{31}$$

Multiple confirmed symptoms provide the bonus

$$b_{\mathrm{multi}}(v) = 0.04\, \mathbf{1}[|L_v| > 1]. \tag{32}$$

The fused confidence is therefore

$$c_{\mathrm{f}}(v) = \mathrm{clip}_{[0,1]}\left(c_{\mathrm{mix}}(v) + b_{\mathrm{multi}}(v)\right). \tag{33}$$

A replayable node is eligible when it has a confirmed category and

$$c_{\mathrm{f}}(v) \geq \theta, \qquad \theta = 0.50. \tag{34}$$

Algorithm 4 implements the online intervention flow: nodes are processed in observed order, and the first eligible node whose fused score meets the threshold triggers replay and live repair. These quantities are ordinal evidence scores, not calibrated probabilities.

Algorithm 4 Online Suspicious-Node Intervention

Require: Replay bundle $\mathcal{S}$, task T, candidate set $\mathcal{C}$, threshold θ
Ensure: Regenerated trace and native-evaluator outcome, or no intervention

for each $(v, L_v, E_v, \widetilde{c}_{\mathrm{LLM}}(v)) \in \mathcal{C}$ in observed order **do**
  Derive local evidence, propagation, node-type, and position features
  Compute $S_{\mathrm{r}}(v)$ using Equation 26
  Compute $S_{\mathrm{s}}(v)$ using Equation 30
  Compute $c_{\mathrm{f}}(v)$ using Equation 33
  if v is a replayable LLM boundary and $c_{\mathrm{f}}(v) \geq \theta$ **then**
    Build $\Delta^\star$ from the target ID, L_v, and rule-derived E_v
    Exclude the judge rationale and all evaluator-derived information
    $\widehat{\tau} \leftarrow$ Replay$(\mathcal{S}, \mathrm{selective}, v, \Delta^\star)$ via Algorithm 2
    $y \leftarrow$ NativeEvaluator$(\widehat{\tau})$
    return $(\widehat{\tau}, y)$
  end if
end for
return no intervention

Coefficients encode modeling priorities and were not optimized using validation repair outcomes; $\theta = 0.50$ is a coverage-oriented gate rather than a globally optimal threshold.

F.3 Intervention Prompts

Suspicious-Node Intervention. The controller instantiates the following template: *Reconsider the decision at target [target identifier]. Confirmed runtime symptoms: [categories]. Trace evidence: [rule-derived evidence]. Revise this step so that the remaining execution addresses the evidence and the task requirements, and regenerate any affected downstream actions or artifacts.* The free-form semantic-judge rationale, reference answer, evaluator verdict, and evaluator-derived evidence are not included.

Random-Node and Last-Node Intervention. Both controls receive the same generic template: *Recheck the task requirements and the selected step. Correct any issue at this step, then regenerate the affected downstream actions or artifacts.* The template contains no symptom category or trace evidence.

F.4 Metrics and Statistical Procedure

Let $R_i^{(m)}$ indicate whether method m repairs failed case i. For the single-intervention node-level meth-

ods,

$$\text{pass@1}(m) = \frac{1}{N}\sum_{i=1}^{N} R_i^{(m)}. \tag{35}$$

Infrastructure and framework errors remain non-repairs; the case is the inferential unit. Statistical comparisons follow Appendix C.

F.5 Threshold Interpretation

Selected-node scores in the primary 536-case evaluation set have unrounded minimum, 25th-percentile, median, 75th-percentile, and maximum values of 0.5003744, 0.6900606, 0.8910112, 0.9360944, and 1.0000000, respectively; these round to 0.50, 0.69, 0.89, 0.94, and 1.00. In a post-hoc threshold-as-abstention analysis that keeps each historical rank-0 target and its observed repair outcome fixed, $\theta = 0.50$ retains 536/536 cases (100.00% coverage), including 108 successful repairs, for $108/536 = 20.15\%$ conditional repair. At $\theta = 0.95$, 78/536 cases are retained (14.55% coverage), including 19 successful repairs, for $19/78 = 24.36\%$ conditional repair. This analysis neither reranks candidate nodes nor reruns repairs at alternative targets.

We interpret 0.50 as a minimum evidence gate. The configuration is practically effective but neither the threshold nor individual coefficients are claimed to be globally optimal. Systematic optimization and component-level ablation remain limitations.

Note on scoring parameters. The implemented selection coefficients, in the order used above, are

$$\begin{aligned}&(\alpha_{\mathrm{r}}, \alpha_{\mathrm{l}}, \alpha_{\mathrm{u}}, \alpha_{\mathrm{n}}, \alpha_{\mathrm{a}}, \alpha_{\mathrm{p}}, \alpha_{\mathrm{f}}, \alpha_{\pi})\\ &\quad = (0.48, 0.32, 0.12, 0.04, 0.08, 0.28, 0.18, 0.12).\end{aligned} \tag{36}$$

Here, $\alpha_{\mathrm{u}} = 0.12$ multiplies the raw confidence $c_{\mathrm{r}}(v)$, which may include propagated evidence, and $\alpha_{\mathrm{n}} = 0.04$ is multiplied by $\min(2, |L_v|)$, so its contribution is capped at 0.08. The repairability coefficients are

$$\begin{aligned}&(\beta_0, \beta_{\mathrm{l}}, \beta_{\mathrm{m}}, \beta_{C_2}, \beta_{C_4}, \beta_{\mathrm{p}}, \beta_{\pi})\\ &\quad = (0.20, 0.26, 0.08, 0.05, 0.07, 0.30, 0.18),\end{aligned} \tag{37}$$

with the node-type prior

$$\eta(v) = \begin{cases} 0.42, & \text{type}(v) = \texttt{agent\_action},\\ 0.36, & \text{type}(v) = \texttt{tool\_call},\\ 0.24, & \text{type}(v) = \texttt{tool\_result},\\ -0.18, & \text{type}(v) = \texttt{final\_result},\\ 0, & \text{otherwise.}\end{cases} \tag{38}$$

The penalties appear with positive coefficients in P_{r} and P_{s} and are subtracted by Equations 26 and 30. The fusion weights 0.55 for semantic-judge confidence and 0.45 for the structured selection score are exact, as is the separate +0.04 bonus for more than one confirmed symptom. These values are hard-coded in `code/src/run_rq3_node_llm_judge.py` and enumerated in `if/data/rq3_node_judge_hyperparameters/tables/hyperparameter-inventory.csv`. They are manually specified heuristic design weights; no saved validation-set optimization or alternative-weight sweep exists, so we do not describe them as fitted, calibrated, or empirically optimal.

F.6 Semantic Boundary of Suspiciousness

To formalize the semantic-boundary audit summarized in the main paper, define

$$\begin{aligned}\mathcal{E}_{\mathrm{f}} = \{&\texttt{failed}, \texttt{failure}, \texttt{error},\\ &\texttt{timeout}, \texttt{timed\_out}\},\end{aligned} \tag{39}$$

The node-level error indicator is

$$S_{\mathrm{node}}(v) = \mathbf{1}[\texttt{v.error} \neq \emptyset]. \tag{40}$$

The tool-result error indicator is

$$S_{\mathrm{tool}}(v) = \mathbf{1}[\texttt{v.tool\_result.error} \neq \emptyset], \tag{41}$$

and the tool-status indicator is

$$S_{\mathrm{status}}(v) = \mathbf{1}[\texttt{v.tool\_result.status} \in \mathcal{E}_{\mathrm{f}}]. \tag{42}$$

The final runtime-failure signal combines the three indicators:

$$S(v) = \max\left\{S_{\mathrm{node}}(v), S_{\mathrm{tool}}(v), S_{\mathrm{status}}(v)\right\}. \tag{43}$$

G Traceable Gold Failure Examples

Corpus anchor and selection protocol

The gold ledger contains **536 finalized, human-adjudicated failure cases**: 462 from WebArena-Verified and 74 from AssistantBench. The corresponding multi-agent-system distribution is 171 AG2, 184 CrewAI, and 181 Magentic-One cases. We select one *trace-verified* example from every observed primary symptom category: C1: 125/536 C2: 69/536 C3: 203/536 C4: 139/536. Thus, the examples cover all four

symptom categories observed in the primary distribution. Selection is stratified by category rather than by favorable outcome. Each example reports the immutable case, annotation, task, and node identifiers; a field-preserving excerpt of the recorded trace; the evidence-supported localization; and the relative path of the complete raw-log bundle. Ellipses inside excerpts remove only payload that is irrelevant to the stated attribution; the raw bundle remains the authoritative record.

GOLD–C1 Standard Clear Positive ✓

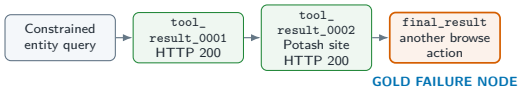
TRACE-VERIFIED

Case / annotation	`40fe6e29c9209dfd` / `9f6c08b2c4fe9155`
Task ID	`assistantbench_dev_9fce12bb25f697d6`
Source / MAS	AssistantBench development split / AG2
Trace size	15 nodes; 3 recorded API interactions
Gold failure node	`assistantbench_dev_9fce12bb25f697d6:attempt_000:snapshot:31ff9ace4b7fbae1:final_result`
Raw-log bundle	`data/initial_failure_logs/ag2/assistantbench_dev_9fce12bb25f697d6/attempt_000`

Task prompt

Which supermarkets within 2 blocks of Lincoln Park in Chicago have ready-to-eat salad for under $15?

Recorded execution path



Field-preserving raw trace excerpt

```
case_id = "40fe6e29c9209dfd"
annotation_id = "9f6c08b2c4fe9155"

failure_node_id =
"assistantbench_dev_9fce12bb25f697d6:attempt_000
      :
 snapshot:31ff9ace4b7fbae1:final_result"

tool_result_0001.status = "success"
tool_result_0001.http_status = 200

tool_result_0002.status = "success"
tool_result_0002.http_status = 200
tool_result_0002.url =
"https://www.potashmarkets.com/"

final_result =
{"action":"browse",
 "url":"https://www.potashmarkets.com/",
 "reason":"Find Potash Markets locations and
     addresses
           near Lincoln Park."}

evaluator.reference_items =
["Potash Markets - Clark Street"]

evaluator.missing_reference_items =
["Potash Markets - Clark Street"]
```

Attribution

C1 Task-constraint violation / incomplete resolution.

Erroneous node. The terminal `final_result` node is the first node at which the trajectory irreversibly fails to satisfy the requested output contract.

Error committed. Instead of returning the name of a qualifying supermarket, the node returns another browsing action. The trajectory therefore terminates before converting successfully retrieved evidence into the requested entity-level answer.

Supporting trace evidence. Both preceding tool results succeeded with HTTP status 200, and the second result reached the Potash Markets website. This rules out an access failure at the attributed node. The evaluator independently records *Potash Markets – Clark Street* as both a reference item and a missing item. The failure is consequently an incomplete answer, not an unavailable-resource failure.

Table 11: Four stratified gold examples. "Prevalence" is computed from the 536-case finalized ledger by primary category. The displayed examples are qualitative trace evidence, not estimates of model accuracy.

Stratum	Selection role	Case ID	Benchmark / MAS	Gold node kind	Prevalence
C1	Standard clear positive	`40fe6e29c9209dfd`	AssistantBench / AG2	`final_result`	23.32%
C2	Complex multi-hop context	`1aaa0ae52cbcf325`	WebArena / AG2	`tool_result`	12.87%
C3	Operational low-signal noise	`06845b3bc9dad059`	WebArena / AG2	`tool_result`	37.87%
C4	Plan–action inconsistency	`11772545c348fb2a`	WebArena / CrewAI	`agent_action`	25.93%

GOLD–C2 Multi-Target and Complex Context ✓

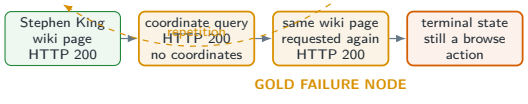
TRACE-VERIFIED

Case / annotation	`1aaa0ae52cbcf325` / `e5b605fd443d173b`
Task ID	`webarena_verified_hard_267`
Source / MAS	WebArena verified-hard split / AG2
Trace size	15 nodes; 3 recorded API interactions
Gold failure node	`webarena_verified_hard_267:attempt_000:snapshot:9ad08821a7a42cde:tool_result_0002`
Raw-log bundle	`data/initial_failure_logs/ag2/webarena_verified_hard_267/attempt_000`

Task prompt

Get the relation ID of the closest national park to the hometown of Stephen King and the time to drive there. Return a list of objects with keys ***`relation_id`*** *(integer) and* ***`duration`*** *(in HH:MM:SS format) only, without any additional details. Use the OSRM direction service and the provided wiki to look up any needed information, and search both source and destination by coordinates from the place official page on the wiki.*

Recorded execution path



Field-preserving raw trace excerpt

```
case_id = "1aaa0ae52cbcf325"
annotation_id = "e5b605fd443d173b"

failure_node_id =
"webarena_verified_hard_267:attempt_000:
 snapshot:9ad08821a7a42cde:tool_result_0002"

tool_call_0000.url =
"https://en.wikipedia.org/wiki/Stephen_King"

tool_result_0000.status = "success"
tool_result_0000.http_status = 200
tool_result_0000.content_length = 889638

tool_call_0001.query.prop = "coordinates"
tool_result_0001.status = "success"
tool_result_0001.http_status = 200
tool_result_0001.response =
{"batchcomplete":"",
 "query":{"pages":{"26954":
   {"pageid":26954,"ns":0,"title":"Stephen King
     "}}}}

tool_call_0002.url =
"https://en.wikipedia.org/wiki/Stephen_King"

tool_result_0002.status = "success"
tool_result_0002.http_status = 200
tool_result_0002.content_length = 889638

final_result.action = "browse"
final_result.url =
"https://en.wikipedia.org/wiki/Stephen_King"

evaluator.expected =
[{"duration":"01:33:00","relation_id":2176999}]
```

Attribution

C2 Repetition, loop, or progress deadlock.

Erroneous node. The gold locus is `tool_result_0002`, which completes a repeated request for the same Stephen King page after the coordinate query has already failed to provide coordinates.

Error committed. The task requires a multi-hop chain: hometown identification, source coordinates, nearest-national-park identification, destination coordinates, OSRM routing, relation-ID lookup, and schema-constrained formatting. Instead of advancing to a new subgoal, the trajectory returns to the previously retrieved page.

Supporting trace evidence. The first and third requests use the same Wikipedia URL and both return HTTP 200 with the same recorded content length of 889638. The intermediate coordinate query also returns HTTP 200, but its response contains page identity fields and no coordinates. The final state remains a browsing action, whereas the evaluator expects `relation_id=2176999` and `duration=01:33:00`. Successful HTTP status therefore does not explain the failure; the decisive problem is lack of progress.

GOLD–C3 Incomplete or Low-Signal Operational Context ✓

TRACE-VERIFIED

Case / annotation	`06845b3bc9dad059` / `fadb3bf17df2786e`
Task ID	`webarena_verified_hard_320`
Source / MAS	WebArena verified-hard split / AG2
Trace size	13 nodes; 3 recorded API interactions
Gold failure node	`webarena_verified_hard_320:attempt_000:snapshot:9ad08821a7a42cde:tool_result_0000`
Raw-log bundle	`data/initial_failure_logs/ag2/webarena_verified_hard_320/attempt_000`

Task prompt

How much refund should I expect from my orders canceled, if any, in February 2023, including the shipping fee? Return the value as a number only, without any additional details.

Recorded execution path

Field-preserving raw trace excerpt

```
case_id = "06845b3bc9dad059"
annotation_id = "fadb3bf17df2786e"

failure_node_id =
"webarena_verified_hard_320:attempt_000:
 snapshot:9ad08821a7a42cde:tool_result_0000"

tool_call_0000.url = "__SHOPPING__"
tool_result_0000.status = "failed"
tool_result_0000.error.type = "MissingSchema"
tool_result_0000.error.message =
"Invalid URL '__SHOPPING__': No scheme supplied.
 Perhaps you meant https://__SHOPPING__"

tool_call_0001.url = "https://__SHOPPING__"
tool_result_0001.status = "failed"
tool_result_0001.error.type = "ConnectionError"
tool_result_0001.error.message =
"... Failed to resolve '__shopping__' ..."

final_result = 58.99
evaluator.expected = 406.53
```

Attribution

C3 Unresolved execution, access, or environmental context.

Erroneous node. The earliest decisive failure is `tool_result_0000`, which receives an unresolved environment placeholder instead of an executable shopping-site URL.

Error committed. The interaction cannot enter the target application. The initial call fails because the placeholder has no URL scheme; adding `https://` does not repair the missing environment binding and instead produces a name-resolution failure.

Supporting trace evidence. The recorded exceptions explicitly identify `MissingSchema` followed by `ConnectionError`. No order history is observed before the trajectory emits 58.99, which differs from the evaluator value of 406.53. The bundle itself is structurally complete; the low-signal condition lies in the unresolved execution context captured by the trace, not in missing provenance files.

GOLD–C4 Plan–Action Inconsistency ✓

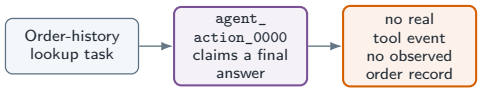
TRACE-VERIFIED

Case / annotation	`11772545c348fb2a` / `f480002d4509ee1e`
Task ID	`webarena_verified_hard_204`
Source / MAS	WebArena verified-hard split / CrewAI
Trace size	5 nodes; 1 recorded API interaction
Gold failure node	`webarena_verified_hard_204:attempt_000:snapshot:9ad08821a7a42cde:agent_action_0000`
Raw-log bundle	`data/initial_failure_logs/crewai/webarena_verified_hard_204/attempt_000`

Task prompt

Get the product name and final price, ordered from low to high, of the most recent completed order. Return a list of objects with keys `name` *and* `price` *only, without any additional details.*

Recorded execution path



GOLD FAILURE NODE

Field-preserving raw trace excerpt

```
case_id = "11772545c348fb2a"
annotation_id = "f480002d4509ee1e"

failure_node_id =
"webarena_verified_hard_204:attempt_000:
 snapshot:9ad08821a7a42cde:agent_action_0000"

agent_action_0000 =
Final answer:
[{"name":"Product B","price":9.99},
 {"name":"Product A","price":19.99}]

Evidence needed:
To obtain the actual data, I would browse to the
__SHOPPING_ADMIN__ page...

evaluator.has_real_tool_event = false
evaluator.observed_tool_result = {}

evaluator.expected =
[{"name":"Ida Workout Parachute Pant","price
     ":38.4},
 {"name":"Proteus Fitness Jackshirt","price
     ":45.0}]
```

Attribution

C4 Plan–action–outcome inconsistency.

Erroneous node. The failure is attributed to `agent_action_0000`, where the agent simultaneously presents a purported final answer and admits that it would still need to browse the shopping administration page to obtain the actual data.

Error committed. The output is superficially plausible: it is valid JSON-like data, contains the requested keys, and is sorted by price. However, the action's own explanation contradicts its epistemic status. It claims specific products and prices before observing any order record.

Supporting trace evidence. The evaluator records `has_real_tool_event=false` and an empty `observed_tool_result`. The fabricated names and prices also differ from the expected products and prices. Thus, the error is not merely an incorrect value or formatting defect; it is a mismatch between the node's claimed answer and the evidence-gathering action that the same node says remains necessary.

H Supplementary Artifact Contents

The code and data supplement includes the complete source code of SymTrace and the complete SymFail dataset used in this study.