Fault Localization in LLM-Based Multi-Agent Systems

Scope-Guided LLM Judging for Responsible-Agent and Failure-Step Attribution

Yavor Pachedzhiev[1]

Supervisors: Dr. Burcu Kulahcioglu Ozkan[1], **Dr. Annibale Panichella**[1], **M.Sc. Zahra Seyedghorban**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Yavor Pachedzhiev
Final project course: CSE3000 Research Project
Thesis committee: Dr. Burcu Kulahcioglu Ozkan, Dr. Annibale Panichella, M.Sc. Zahra Seyedghorban, Dr. Matthijs Spaan

Abstract

LLM-based multi-agent systems often produce long execution traces with agent messages, tool outputs, intermediate decisions, and final responses. When a task fails, the failed outcome usually does not show which agent caused the failure or which earlier step introduced it. This paper treats fault localization as failure attribution: predicting the responsible agent and the decisive failure step in a failed multi-agent trace. It compares a direct whole-trace baseline with two-stage scope-guided judging on the Hand-Crafted subset of the Who&When benchmark. In the direct baseline, one LLM judge receives the full trace and predicts both labels. In the scope-guided methods, a first-stage selector chooses a small set of reference steps, and the same final judge predicts the labels from the full trace plus those selected steps. The experiments show that scope guidance is not generally beneficial. Generic LLM scope selection improves selected-scope Hit@5 over random selection, but does not improve final attribution over direct whole-trace judging. The source-candidate-pool selector gives the best responsible-agent, failure-step, and joint attribution accuracy, but the improvement is modest and requires more than four times the mean token cost of direct whole-trace judging. Overall, scope guidance helps only when the selected steps point the judge toward earlier source-level evidence. Direct whole-trace judging remains a strong lower-cost baseline.

1 Introduction

LLM-based multi-agent systems solve tasks through several interacting language-model agents. Such systems have been studied in role-based software-development workflows and in broader analyses of multi-agent LLM failures [1; 2]. A run may include planning, tool calls, web browsing, file inspection, intermediate decisions, and a final response. When such a run fails, the failed task outcome usually does not show where the failure entered the trace. The visible failure may appear late, while the earlier cause may be an incorrect observation, an accepted but incomplete result, a missed verification step, or a poor handoff between agents.

Recent work frames this problem as failure attribution: identifying which agent caused a failed multi-agent run and when the decisive error occurred [3]. In this paper, fault localization is operationalized in the same way, by predicting both the responsible agent and the decisive failure step in a failed execution trace. This makes the localization result more useful for debugging. Instead of only knowing that a run failed, a developer receives a candidate agent and trace step to inspect when revising prompts, roles, tool use, or verification logic.

A simple baseline is direct whole-trace judging. In this setup, one LLM judge receives the full failed trace and predicts both the responsible agent and the decisive failure step. This keeps all context available, but the judge must search through the whole trace at once. In long traces, the decisive step may be surrounded by many plausible but non-decisive steps, including later symptoms of the same failure.

Motivated by scope delineation before localization [4], this paper studies whether a two-stage scope-guided setup can improve this attribution task. In the scope-guided setup, Stage 1 selects a small set of trace steps as reference steps. Stage 2 uses the same final LLM judge as the direct baseline, but gives it both the full trace and the selected reference steps. The selected steps guide the judge's attention, but they do not replace the full trace and they do not restrict the final prediction.

1.1 Research Questions

The main research question is:

> **RQ:** Can a two-stage LLM judging pipeline improve failure attribution in multi-agent LLM execution traces by using selected trace steps to guide the prediction of the responsible agent and decisive failure step?

This question is divided into three subquestions:

> **SQ1 – Scope quality:** How accurately do different scope-selection methods recover the gold decisive failure step within their selected trace scope?
>
> **SQ2 – Attribution accuracy:** How does scope-guided judging affect responsible-agent, failure-step, and joint attribution accuracy compared with direct whole-trace judging?
>
> **SQ3 – Practical trade-offs:** What token-cost, output-validity, and error-propagation trade-offs arise from introducing scope selection before final attribution?

The experiments use the Hand-Crafted subset of the Who&When benchmark, which contains failed multi-agent execution traces annotated with the failure-responsible agent and decisive error step [3]. This paper refers to this annotated step as the decisive failure step. The subset contains 58 failed traces and 2,993 trace steps in total, with a mean length of 51.6 steps and a maximum length of 130 steps.

The comparison includes direct whole-trace judging and four scope-guided methods: random scope selection, generic LLM scope selection, an Expertise-Assisted Scope Delineation (EASD)-inspired adapter based on overstep and loop patterns from scope-delineation work [4], and a source-candidate-pool selector developed for this project. All scope-guided methods use the same final attribution judge as the direct baseline, so the comparison tests whether selected reference steps help the same judge make better attribution decisions.

The main finding is conditional: scope guidance helps only for the source-candidate-pool selector, not as a general effect of adding selected reference steps. Generic LLM scope selection improves selected-scope Hit@5 over random selection, but does not improve final attribution over direct whole-trace judging. The source-candidate-pool selector gives the best responsible-agent, failure-step, and joint accuracy, but at substantially higher token cost. Direct whole-trace judging therefore remains a strong low-cost baseline.

1.2 Contributions

This work makes three contributions.

- It evaluates scope-guided LLM judging as a two-stage approach to failure attribution in multi-agent execution traces.
- It separates scope quality from final attribution accuracy by measuring both selected-scope Hit@5 and final responsible-agent, failure-step, and joint accuracy.
- It compares random, generic LLM, EASD-inspired, and source-candidate-pool scope selection, showing that scope quality alone does not guarantee better final attribution and that the strongest method trades modest accuracy gains for substantially higher token cost.

2 Background and Task Definition

2.1 LLM-Based Multi-Agent Systems and Execution Traces

LLM-based multi-agent systems organize several language-model agents into a shared workflow. Agents may have different roles, tools, or responsibilities, such as coordination, web browsing, file inspection, coding, testing, or final answer generation. Such systems have been studied in role-based software-development workflows and in broader analyses of multi-agent LLM failures [1; 2].

A failed run is represented as an execution trace: an ordered record of the system's behavior. Each step in the trace has an index, an actor, and textual content such as an agent message, a tool result, an intermediate decision, or a final response. The trace therefore preserves the process that led to the failed outcome.

However, the trace can also make attribution difficult. Later steps may be consistent with an already-corrupted state of the execution, even if they are not the source of the failure. A final response or late repeated attempt may expose the failure, while the earlier cause may be a wrong observation, a missed constraint, an accepted but incomplete result, or a poor handoff between agents.

Recent work on agentic trace diagnosis makes a similar distinction between outcome-level failure and process-level causes. AgentRx studies failed trajectories annotated with a critical failure step and uses constraint-violation logs as evidence for localization [5]. AgentPex focuses on specification-based trace evaluation and shows that outcome-only scoring can miss procedural violations inside an agent trace [6].

2.2 Failure Attribution

Failure attribution identifies the part of a failed execution trace that is most responsible for the failed outcome. In this paper, the attribution task has two outputs. The first is the *responsible agent*: the agent whose behavior most directly caused the failed outcome. The second is the *decisive failure step*: the earliest consequential step where the mistake was introduced or made unavoidable.

This definition separates a failure source from a downstream symptom. A *failure source* is an earlier step that introduces faulty information, accepts weak evidence, misses a required constraint, or commits the system to a failing path.

Step	Actor	Abridged trace event	Role in the failure
1	User	Asks for a martial arts class near the New York Stock Exchange that is open between 7–9 p.m. and within a five-minute walk.	Task constraints.
2	Orchestrator	Instructs the WebSurfer to search for suitable classes and check opening hours and walking distance.	Delegation.
3	WebSurfer	Reports a class with evening availability, but overlooks that it is around 12–15 minutes away from the requested location.	Missed constraint.
4	Orchestrator	Accepts the WebSurfer result as satisfying the constraints and proceeds to prepare the final answer.	Commitment to faulty evidence.
5	Orchestrator	Gives the class as the final answer, although it violates the walking-distance constraint.	Visible failure.

Table 1: Abridged example of a failure source and a downstream symptom. The final answer exposes the failure, but the decisive failure step may occur earlier when a constraint is missed or accepted without verification.

A *downstream symptom* is a later step where the failure becomes visible, such as an incorrect final answer or an unresolved repeated attempt. The distinction matters because a judge that sees a long trace may focus on visible symptoms rather than on the earlier step that caused them.

2.3 Failure Sources and Downstream Symptoms

Table 1 gives an abridged example of the difference between a failure source and a downstream symptom. The user asks for a martial arts class near the New York Stock Exchange that is open in the evening and within a five-minute walk. The final answer violates the walking-distance constraint, but the failure is caused earlier.

In this example, step 5 makes the failure visible. However, step 3 introduces faulty evidence by missing the walking-distance constraint, and step 4 commits to that evidence without further verification. This illustrates why the attribution target is not simply the last wrong message, but the earlier step that introduced the error or committed the system to the failing path.

2.4 Scope Selection and Related Attribution Work

Long traces create a search problem for failure attribution. A direct judge receives the full trace and must identify the responsible agent and decisive failure step from all available steps. Scope selection instead tries to identify a smaller set of trace steps that deserve special attention before the final attribution decision is made.

This idea follows the motivation of scope delineation before localization: selecting a relevant part of a long trace may

make later failure analysis easier [4]. In this paper, selected steps are used as reference steps for the final judge. They guide attention, but they do not replace the full trace and they do not restrict which step the judge may choose.

Other recent attribution methods also avoid treating the full execution log as an unstructured flat sequence. ECHO uses hierarchical context representation and consensus analysis [7], CHIEF reconstructs trajectories as hierarchical causal graphs [8], and FALAT frames attribution as dependency-guided search over candidate failure steps [9]. These methods are not evaluated in this paper. They are relevant because they share the broader motivation that failure attribution often requires reducing or structuring the search over long traces.

2.5 Task Definition

Let $T = (x_1, \ldots, x_n)$ be a failed execution trace, where each step x_i has an index i, an actor, and textual content. Let A be the set of system agents in the trace. The gold attribution pair is (a^*, s^*), where $a^* \in A$ is the responsible agent and $s^* \in \{1, \ldots, n\}$ is the decisive failure step. The responsible agent must be one of the system agents rather than the user.

For direct whole-trace judging, the input is the full failed trace T, and the output is a predicted attribution pair $(\hat{a}, \hat{s})$. For scope-guided judging, a first stage selects a set of reference steps $S \subseteq \{1, \ldots, n\}$. A second stage receives both the full trace T and the selected reference steps S, and then predicts the attribution pair $(\hat{a}, \hat{s})$.

The gold labels used in this paper follow the responsible-agent and decisive-error-step annotations from the Who&When benchmark [3]. This paper refers to the decisive error step as the decisive failure step. The dataset split, prompt construction, evaluation metrics, token accounting, and output-validity checks are described in Section 4.

3 Methods

This section describes the five judging methods evaluated in this paper. All methods receive a failed execution trace and produce the same output: a predicted responsible agent and a predicted decisive failure step. The methods differ in whether they use a Stage 1 scope selector before final attribution. Direct whole-trace judging does not use a selector. The four scope-guided methods first select reference steps and then pass those steps to the same final attribution judge used by the direct baseline.

All methods are prompt-based. The prompts render the trace as a task description, an allowed-agent list, and an ordered list of trace steps. The gold responsible agent, gold decisive failure step, and gold reason are not included in the prompt text.

3.1 Direct Whole-Trace Judging

Direct whole-trace judging is the baseline method. It uses one LLM call. The final attribution judge receives the full failed trace and predicts the responsible agent and decisive failure step directly. No Stage 1 selector is used, and no selected reference steps are added to the prompt.

This baseline is included because it is the simplest way to use an LLM judge for the task. It also gives a direct comparison point for testing whether selected reference steps improve attribution.

3.2 Two-Stage Scope-Guided Judging

The scope-guided methods use two stages. Stage 1 selects trace steps that should receive special attention. Stage 2 uses the shared final attribution judge to predict the responsible agent and decisive failure step.

The selected steps are used as reference steps. They do not replace the full trace. In every scope-guided method, the final judge receives both the full trace and a separate reference block containing the selected steps. The judge may still choose any valid step from the full trace. Therefore, the selected reference steps guide attention, but they are not a hard candidate set.

Figure 1 shows the direct and scope-guided setups.

3.3 Stage 1: Scope Selection Methods

The Stage 1 selector produces the selected reference steps for the final judge. All selectors use a maximum budget of five selected steps. The random selector does not use an LLM call. The other selectors use one or more prompted LLM calls that return step indices. Table 2 summarizes the four selectors.

3.3.1 Random Scope Selection

Random scope selection samples up to five valid trace steps uniformly from the ordered trace. It ignores the task, agent roles, and trace content. The selected steps are sorted before being passed to the final judge. This selector is included to test whether the reference-step format helps when the selected steps are not chosen from trace content.

3.3.2 Generic LLM Scope Selection

Generic LLM scope selection gives the full trace to an LLM selector and asks it to select up to five steps that may contain evidence for the decisive failure. The selector receives no gold labels and uses no specialized failure categories. It represents the simplest content-aware scope selector in the comparison.

3.3.3 EASD Adapter Scope Selection

The EASD adapter is inspired by the Expertise-Assisted Scope Delineation variant from scope-delineation work [4]. It is not a reproduction of that system. It adapts the idea to this paper's trace format and attribution task by using two selector calls.

The first call searches for *overstep*-like behavior. In this adapter, overstep means that an executor or tool-like agent appears to act beyond its expected role, for example by terminating the overall task, declaring global completion, or taking coordination authority. The second call searches for *loop*-like behavior. In this adapter, loop means that the controller repeats similar commands or fails to recover from an earlier problem.

The outputs of the two calls are parsed separately, filtered to valid trace steps and expected item types, deduplicated, and merged into one selected reference-step set. The final selected steps are then passed to the shared final attribution judge.

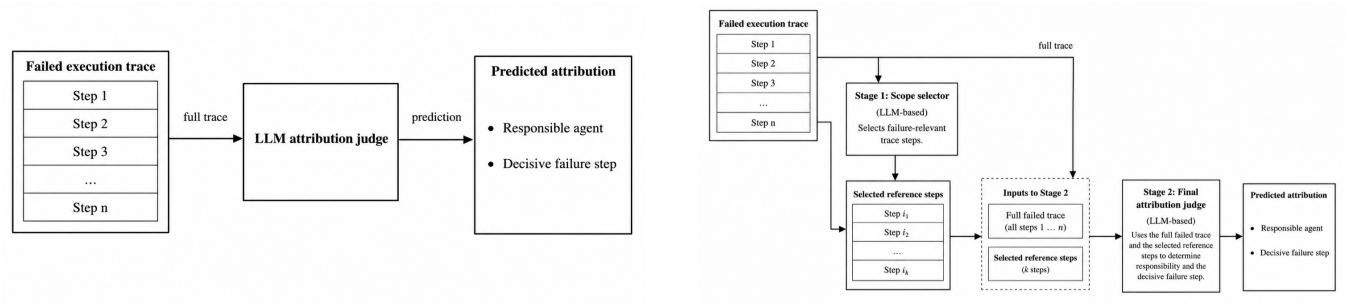

Figure 1: Direct whole-trace judging and two-stage scope-guided judging. In the scope-guided setting, the final judge receives the full trace and an additional reference block containing selected steps.

Selector	Selection principle	Diagnostic bias	Reason for inclusion
Random scope	Samples trace steps uniformly without reading their content.	No diagnostic bias. Any hit is due to chance.	Chance baseline for scope selection.
Generic LLM scope	Asks an LLM to choose steps that may contain evidence for the decisive failure.	Favors steps that are visibly relevant in the full trace.	Simple content-aware selector.
EASD adapter	Uses two pattern-based selector calls inspired by Expertise-Assisted Scope Delineation: overstep and loop.	Favors role overreach, repeated commands, and nearby failure-recovery regions.	Prior-inspired comparison method.
Source-candidate-pool	Builds a candidate pool before selecting final reference steps. Candidate steps are harvested from chunks using task and symptom context, then consolidated globally.	Favors earlier source, commitment, recovery, and termination steps over late symptoms.	Project-developed structured selector.

Table 2: Overview of the Stage 1 scope selectors. The table summarizes how each selector chooses reference steps and what kind of failure evidence it tends to emphasize.

3.3.4 Source-Candidate-Pool Scope Selection

The source-candidate-pool selector is the project-developed Stage 1 method. It is motivated by the difference between downstream symptoms and earlier failure sources. Instead of asking an LLM to choose the final reference steps immediately, the selector first builds a wider pool of possible source or commitment steps and then consolidates that pool.

The selector uses four prompted stages.

Task brief. The first call summarizes the user request and extracts concrete task requirements. This gives the later stages a compact description of the task requirements.

Failure symptoms. The second call identifies visible downstream failure symptoms in the trace. These symptoms are used as context for candidate harvesting. They are not directly used as selected reference steps.

Chunk-level candidate harvesting. The trace is split into overlapping chunks. Each chunk is inspected for possible source or commitment steps. The prompt uses four heuristic candidate types: *source action*, *commitment decision*, *failed recovery*, and *premature termination*. A source action introduces faulty or incomplete information. A commitment decision accepts weak evidence or moves the system onto a failing path. A failed recovery misses an opportunity to correct the problem. A premature termination finalizes or stops while the task is still unresolved. These types are prompt categories used for candidate harvesting, not a formal failure taxonomy.

Global consolidation. Candidate steps from all chunks are deduplicated into a candidate pool. A final consolidation call selects up to five source candidates from that pool. These selected candidates become the selected reference steps for the final attribution judge. The selected steps are sorted before being passed to the judge, while their priority order is kept only for diagnostics.

Figure 2 summarizes the source-candidate-pool selector at a high level.

3.4 Stage 2: Shared Final Attribution Judge

The same final attribution judge is used for direct whole-trace judging and for all scope-guided methods. In the direct baseline, the judge receives only the full trace. In the scope-guided methods, it receives the full trace plus the selected reference steps from Stage 1.

The final judge predicts one responsible agent and one decisive failure step. The prompt lists the allowed responsible agents and requires a JSON object with a responsible-agent field and a decisive-step field. Since the selected reference steps are guidance rather than a restriction, the predicted decisive step may be inside or outside the selected reference-step set.

Source-candidate-pool selector

Input: failed trace, user task, agent list **Output:** selected reference steps for the final judge

1. **Context construction (LLM calls).** Create a task brief from the user task and identify visible failure symptoms from the trace.
2. **Trace chunking (deterministic).** Split the trace into overlapping chunks so that long traces can be inspected in smaller parts.
3. **Candidate harvesting (LLM call per chunk).** For each chunk, use the task brief and failure symptoms as context to collect possible source, commitment, recovery, and premature-termination candidates.
4. **Candidate-pool construction (deterministic).** Merge the harvested candidates, remove duplicate step indices, and keep a bounded candidate pool.
5. **Global consolidation (LLM call, then deterministic sorting).** Select up to five source candidates from the candidate pool. The selected step indices are then sorted by trace order and used as the reference steps for the final attribution judge.

Figure 2: High-level view of the source-candidate-pool selector. The method builds task and failure context, chunks the trace, harvests possible source candidates from each chunk, constructs a bounded candidate pool, and consolidates that pool into selected reference steps.

4 Experimental Setup

This section describes how the experiments were run. All methods are evaluated on the same traces. All methods use the same final attribution judge. They differ only in whether and how selected reference steps are produced before final attribution.

4.1 Dataset

The experiments use the Hand-Crafted subset of the Who&When benchmark [3]. The name Hand-Crafted refers to the subset released with Who&When, not to data constructed in this project. The subset contains 58 failed execution traces from a hand-crafted LLM-based multi-agent system. Each trace contains the original task, the ordered execution history, the participating agents, the gold responsible agent, and the gold decisive failure step.

The prepared traces preserve the original step ordering and represent each step with a step index, actor, role, optional target, and textual content. The subset contains 2,993 trace steps in total, with a mean of 51.6 steps per trace and a maximum of 130 steps. Gold labels are used only for evaluation and are not included in any prompt.

4.2 Compared Methods

The experiments compare the five methods described in Section 3. Direct whole-trace judging is the main baseline because it performs attribution without a scope-selection stage. Random scope-guided judging is included as a chance baseline for the reference-step format: it tests whether adding selected reference steps helps when the steps are not chosen from trace content. The remaining scope-guided methods use generic LLM scope selection, the EASD adapter, or the source-candidate-pool selector before the shared final attribution judge.

All scope-guided methods use a budget of at most five selected reference steps. Random scope selection uses one fixed seed, 42, and has no Stage 1 token cost because it does not call an LLM. In all scope-guided methods, the selected reference steps are guidance rather than a hard candidate set. The final judge still receives the full trace and may select any valid step.

4.3 Evaluation Metrics

Following scope-delineation evaluation, scope quality is evaluated with Hit@5 [4]. Let s_i^* be the gold decisive failure step for trace i, and let S_i be the selected reference-step set. Hit@5 is defined as:

$$\text{Hit@5} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[s_i^* \in S_i].$$

Hit@5 is reported only for scope-guided methods, since direct whole-trace judging does not select reference steps.

Final attribution quality is evaluated with responsible-agent accuracy, failure-step accuracy, and joint accuracy. Let (a_i^*, s_i^*) be the gold responsible agent and decisive failure step for trace i, and let $(\hat{a}_i, \hat{s}_i)$ be the corresponding prediction. The three metrics are defined as:

$$\text{AgentAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{a}_i = a_i^*],$$

$$\text{StepAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{s}_i = s_i^*],$$

$$\text{JointAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[(\hat{a}_i = a_i^*) \wedge (\hat{s}_i = s_i^*)].$$

4.4 Token Cost and Output Validity

Token usage is tracked because scope-guided methods add one or more model calls before final attribution. For each trace, total token cost is computed as:

$$\text{TotalTokens}_i = \text{ScopeTokens}_i + \text{LocalizationTokens}_i.$$

For direct whole-trace judging and random scope-guided judging, $\text{ScopeTokens}_i = 0$. For generic scope-guided judging, scope tokens come from the single selector call. For the EASD adapter, scope tokens are the sum of the overstep and loop selector calls. For the source-candidate-pool selector, scope tokens include the task-brief call, symptom-summary

call, chunk-level harvesting calls, and global consolidation call.

Output validity is tracked separately from attribution accuracy. Attribution accuracy measures whether a valid prediction matches the gold labels. Output validity measures whether the method produced a usable prediction in the first place. A final attribution output is marked as an error if the model call fails, the response cannot be parsed, the predicted agent is not one of the allowed agents, or the predicted step is not valid for the trace. A scope-selection output is marked as an error if it cannot produce valid selected reference steps.

4.5 Implementation Details

All LLM-based components are implemented as fixed prompt templates with structured JSON outputs. Outputs are parsed automatically into the evaluation format. No model fine-tuning is performed.

The reported runs used `gpt-4.1-mini` with temperature 0. Temperature 0 was used to reduce sampling variation and make the comparisons more reproducible. Final attribution calls used a maximum output length of 300 tokens. The EASD scope-selection calls used a maximum output length of 800 tokens. The source-candidate-pool scope-selection calls used a maximum output length of 1000 tokens.

The rendering code excludes evaluation-only fields such as the gold responsible agent, gold decisive failure step, and gold reason from the prompt text. For scope-guided methods, selected reference steps are normalized to valid trace indices before being passed to the final attribution judge.

The main implementation files, run configuration, and prompt components are summarized in Appendix A.

5 Results

This section reports results on the 58 Hand-Crafted traces. The results are organized around the three subquestions. Overall attribution performance addresses SQ2, scope-selection quality addresses SQ1, and token cost plus output validity addresses SQ3. The final subsection reports additional diagnostics for the source-candidate-pool selector.

5.1 Overall Performance

Table 3 reports the final attribution results. The source-candidate-pool guided method obtains the highest responsible-agent accuracy, failure-step accuracy, and joint accuracy. Compared with direct whole-trace judging, it improves responsible-agent accuracy from 30/58 to 35/58, failure-step accuracy from 14/58 to 17/58, and joint accuracy from 12/58 to 15/58.

The other scope-guided methods do not improve all final attribution metrics. Generic scope-guided judging matches the direct baseline on agent, step, and joint accuracy. Random scope-guided judging and EASD scope-guided judging improve responsible-agent accuracy, but reduce failure-step and joint accuracy.

For SQ2, these results show that scope-guided judging does not generally improve final attribution accuracy. Among the tested methods, only the source-candidate-pool guided method improves all three final attribution metrics over direct whole-trace judging.

5.2 Stage 1 Scope Quality

Table 4 reports scope-selection results. The source-candidate-pool selector has the highest Hit@5, selecting the gold decisive failure step in 26/58 traces. The generic LLM selector is second with 24/58 hits. Random scope selection selects the gold step in 11/58 traces, while the EASD adapter selects it in 13/58 traces.

The scope selectors differ substantially in token cost. Random scope selection has no scope-token cost. Generic LLM scope selection uses one selector call per trace. The EASD adapter and source-candidate-pool selector use multiple selector calls, leading to higher mean scope-token costs.

For SQ1, the source-candidate-pool selector gives the highest selected-scope Hit@5, followed by generic LLM scope selection. The difference between these two selectors is small, while the EASD adapter performs only slightly above random scope selection on this metric.

5.3 Token Cost and Output Validity

Direct whole-trace judging has the lowest mean token cost among methods that use an LLM final judge, with 17,837 mean total tokens per trace. Random scope-guided judging is only slightly more expensive because it adds selected reference steps to the final prompt but does not use an LLM selector.

The content-aware guided methods are more expensive. Generic scope-guided judging uses 38,215 mean total tokens per trace, EASD scope-guided judging uses 54,853, and source-candidate-pool guided judging uses 75,323.

Output-validity errors are rare in these runs. Direct whole-trace judging and generic scope-guided judging have no localization errors. Random scope-guided judging and EASD scope-guided judging each have one localization error. Source-candidate-pool guided judging has two localization errors. All scope selectors produced rows for all 58 traces.

For SQ3, the results show that scope guidance increases token cost, especially when the selector uses multiple LLM calls. The source-candidate-pool guided method gives the best attribution results, but it is also the most expensive method. Output-validity failures are uncommon, but they occur only in scope-guided methods in these runs.

5.4 Source-Candidate-Pool Diagnostics

Table 5 reports additional diagnostics for the source-candidate-pool selector. The gold decisive failure step appears in the intermediate candidate pool for 32/58 traces. After global consolidation, it remains in the final selected reference steps for 26/58 traces. Among traces where the gold step is selected, its mean priority rank is 1.73.

The diagnostics show that the largest loss occurs before global consolidation. The candidate pool misses the gold decisive failure step in 26 traces. Once the gold step enters the pool, it remains in the final selected reference steps in 26/32 cases. This means that six gold steps are lost after entering the candidate pool, while most misses happen during candidate-pool construction.

Method	Agent acc.	Step acc.	Joint acc.	Mean total tokens	Localization errors
Direct whole trace	30/58 (0.517)	14/58 (0.241)	12/58 (0.207)	17,837	0
Random scope guided	33/58 (0.569)	13/58 (0.224)	11/58 (0.190)	18,449	1
Generic scope guided	30/58 (0.517)	14/58 (0.241)	12/58 (0.207)	38,215	0
EASD scope guided	32/58 (0.552)	11/58 (0.190)	9/58 (0.155)	54,853	1
Source-candidate-pool guided	**35/58 (0.603)**	**17/58 (0.293)**	**15/58 (0.259)**	75,323	2

Table 3: Final attribution results on the Who&When Hand-Crafted subset. Agent accuracy, step accuracy, and joint accuracy are computed over all 58 traces. Mean total tokens include both scope-selection and final-attribution calls where applicable.

Scope selector	Hit@5	Mean selected steps	Mean scope tokens
Random scope	11/58 (0.190)	5.00	0
Generic LLM scope	24/58 (0.414)	4.31	17,692
EASD adapter	13/58 (0.224)	3.60	36,778
Source-candidate-pool	**26/58 (0.448)**	4.03	56,159

Table 4: Stage 1 scope-selection results. Hit@5 measures whether the selected reference steps contain the gold decisive failure step. Random scope selection has no scope-token cost because it does not use an LLM call.

6 Discussion

6.1 Interpreting the Effect of Scope Guidance

The answer to the research question is conditional. The results do not show that adding selected reference steps generally improves failure attribution. The important factor is what kind of evidence the selected steps provide. If the selected steps mainly point to visible or apparently relevant parts of the trace, they may not help the final judge identify the earlier source of the failure.

This explains the difference between the guided methods. Generic LLM scope selection improves selected-scope Hit@5 over random selection, but this does not improve final attribution. The EASD adapter also does not improve failure-step or joint accuracy over direct whole-trace judging. In contrast, the source-candidate-pool selector is designed to search for earlier source, commitment, recovery, and termination steps. It is therefore the only guided method that improves all final attribution metrics in this experiment.

The source-candidate-pool diagnostics suggest that candidate-pool construction is the main bottleneck. When the gold decisive failure step enters the candidate pool, global consolidation usually keeps it. When the gold step is missed during candidate harvesting, later stages cannot recover it. This means that improving the chunk-level candidate harvesting stage would likely matter more than only changing the final consolidation step.

6.2 Why Scope Quality Does Not Fully Determine Attribution Accuracy

Scope Hit@5 and final attribution accuracy measure different things. Hit@5 only checks whether the gold decisive failure step appears somewhere in the selected reference steps. It does not measure whether that step is easy to identify as decisive, whether other selected steps distract the judge, or whether the final judge uses the selected evidence correctly.

The final task is also harder than selecting a relevant trace region. The judge must link the decisive failure step to the responsible agent, and the responsible agent does not always have to be the actor at the decisive step. A selected reference set may therefore contain the correct step while the final judge still chooses a later symptom, another step in the same causal chain, or the wrong responsible agent.

This means that scope selection is only one part of the attribution problem. Selected reference steps can guide attention, but they do not determine the final prediction because the judge still receives the full trace and may choose any valid step.

6.3 Practical Value Versus Cost

The practical value of scope-guided judging depends on whether the additional cost is worth the improvement. Direct whole-trace judging remains a strong lower-cost baseline because it uses only the final attribution call and still performs competitively on this dataset.

The source-candidate-pool method gives the best attribution results and useful intermediate diagnostics, but it is also the most expensive method. This makes it more suitable for analysis settings where better attribution and insight into the selector's behavior are worth the extra token cost. For lower-cost settings, direct whole-trace judging remains the more practical choice.

Overall, the results support a limited trade-off. Scope guidance can help when the selected reference steps point toward source-level evidence, but it does not help equally across all selector designs tested here.

Diagnostic	Candidate pool hit	Selected Hit@5	Priority Hit@1	Priority Hit@3	Mean gold rank
Source-candidate-pool	32/58 (0.552)	26/58 (0.448)	13/58 (0.224)	24/58 (0.414)	1.73

Table 5: Source-candidate-pool diagnostics. Candidate pool hit measures whether the gold decisive failure step appears in the intermediate candidate pool before consolidation. Selected Hit@5 measures whether it remains in the final selected reference steps. Mean gold rank is computed only for traces where the gold step is selected.

7 Threats to Validity and Responsible Research

7.1 Dataset and Scope

The experiments use only the Hand-Crafted subset of the Who&When benchmark. This makes the evaluation controlled, because every trace has a gold responsible agent and decisive failure step, but it also limits how far the results can be generalized. The subset contains 58 failed traces from one benchmark setting. Other multi-agent systems may have different agent roles, less structured logs, missing tool outputs, or failures that do not have one clear decisive step.

The results should therefore be read as evidence for this benchmark setting, not as a general claim about all LLM-based multi-agent systems. A useful next step would be to evaluate the same methods on other failure-attribution datasets, such as AEGIS or CORRECT [10; 11].

7.2 Prompt, Model, and Randomness

All LLM-based methods use fixed prompts and one model configuration. The results may change with a different model, different prompt wording, different output parser, or different decoding settings. Temperature 0 was used to reduce sampling variation, but it does not guarantee that every API call will always give the same output.

The EASD adapter is also a limitation. It adapts overstep and loop ideas from scope-delineation work to this task, but it is not a full reproduction of the original method. Random scope selection was run with one fixed seed. This makes the experiment reproducible, but a stronger random baseline would average over several seeds.

7.3 Evaluation Limits

The evaluation uses exact-match accuracy. This is easy to reproduce, but it is strict. A prediction is counted as wrong even if it selects a nearby step, a later symptom of the same failure, or another contributing step in the same causal chain. Scope Hit@5 has a similar limitation: it only checks whether the gold step is included in the selected reference steps, not whether the selected steps are actually helpful to the final judge.

The dataset is also small, so the results should be interpreted carefully. A difference of a few correct predictions changes the reported accuracy. Token cost is reported using the token counts from this model setup, so the exact costs may differ for other providers, models, or prompt formats.

7.4 Reproducibility and Use of AI

The experiments use benchmark traces rather than private deployment logs. The source code, notebooks, prepared data files, outputs, and exact prompt text are provided in the replication package.[1] Appendix A lists the main code files, run configuration, and prompt components.

AI tools were used during the project for coding, debugging, and editing text. They helped with writing code, revising sentences, and restructuring some paragraphs. The experiment design, interpretation of the results, final wording, and responsibility for the submitted paper remain with the author.

The attribution methods in this paper should be treated as debugging aids. They can point a reviewer to a likely responsible agent and trace step, but they should not be used as final proof that an agent, developer, or organization is at fault.

8 Conclusion

This paper studied failure attribution as a fault-localization task for LLM-based multi-agent systems. The goal was to identify both the responsible agent and the decisive failure step in failed execution traces. The experiments compared direct whole-trace judging with several two-stage scope-guided methods on the Hand-Crafted subset of the Who&When benchmark.

The results show that scope guidance is not generally beneficial by itself. Generic LLM scope selection improves selected-scope Hit@5 over random selection, but it does not improve final attribution over direct whole-trace judging. Among the tested guided methods, only the source-candidate-pool selector improves responsible-agent, failure-step, and joint attribution accuracy over the direct baseline.

The main conclusion is therefore a limited trade-off. Scope guidance can help when the selected reference steps point the final judge toward earlier source-level evidence, but the improvement is modest and comes with much higher token cost. Direct whole-trace judging remains a strong lower-cost baseline, while the source-candidate-pool method is more useful when better attribution and intermediate diagnostics are worth the extra cost.

References

[1] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development," *arXiv preprint arXiv:2307.07924*, 2024.

[2] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica, "Why do multi-agent llm systems fail?,"

[1] https://github.com/yavor0/Research-Project

in *The Thirty-ninth Annual Conference on Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025.

[3] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu, "Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems," in *Proceedings of the 42nd International Conference on Machine Learning*, vol. 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599, 2025.

[4] K. Sun, W. Li, B. Dong, Y. Lin, J. Zhang, and B. Shi, "Scope delineation before localization: A two-stage framework for enhancing failure attribution in multi-agent systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

[5] S. Barke, A. Goyal, A. Khare, A. Singh, S. Nath, and C. Bansal, "Agentrx: Diagnosing ai agent failures from execution trajectories," *arXiv preprint arXiv:2602.02475*, 2026.

[6] R. K. Sharma, S. Barke, and B. Zorn, "Willful disobedience: Automatically detecting failures in agentic traces," *arXiv preprint arXiv:2603.23806*, 2026.

[7] A. Banerjee, A. Nair, and T. Borogovac, "Where did it all go wrong? a hierarchical look into multi-agent error attribution," *arXiv preprint arXiv:2510.04886*, 2025.

[8] Y. Wang, W. Wu, J. Wang, and Q. Wang, "From flat logs to causal graphs: Hierarchical failure attribution for llm-based multi-agent systems," *arXiv preprint arXiv:2602.23701*, 2026.

[9] M. N. Rafi, M. Ahasanuzzaman, D. J. Kim, Z. Wang, and T.-H. Chen, "Falat: Tracing failures in llm agent trajectories via dependency-guided search," *arXiv preprint arXiv:2606.00765*, 2026.

[10] F. Kong, R. Zhang, H. Yin, G. Zhang, X. Zhang, Z. Chen, Z. Zhang, X. Zhang, S.-C. Zhu, and X. Feng, "Aegis: Automated error generation and attribution for multi-agent systems," *arXiv preprint arXiv:2509.14295*, 2026.

[11] Y. Yu, M. Li, S. Xu, J. Fu, X. Hou, F. Lai, and B. Wang, "Correct: Condensed error recognition via knowledge transfer in multi-agent systems," *arXiv preprint arXiv:2509.24088*, 2025.

A Replication Package and Implementation Summary

This appendix lists the run configuration and the main implementation files used in the experiments. The replication package is linked in Section 7.4. The exact prompts are implemented in the source files listed below.

A.1 Repository Structure

Table 6 shows the main files used to prepare the data, run the methods, and compute the results.

Component	Location in replication package
Dataset preparation	`src/who_when_data.py`, `notebooks/00_prepare_who_when.ipynb`
Trace rendering	`src/rendering.py`
LLM client and token accounting	`src/llm_client.py`
Final attribution prompt and parser	`src/final_localizer.py`
Random and generic scope selection	`src/scope_methods.py`
EASD adapter	`src/easd_adapter.py`, `notebooks/05_easd_scope.ipynb`
Source-candidate-pool selector	`src/source_candidate_pool.py`, `notebooks/06_source_candidate_pool_scope.ipynb`
Result comparison	`notebooks/04_compare_current_methods.ipynb`
Experiment outputs	`outputs/scopes/`, `outputs/localization/`

Table 6: Main files in the replication package.

A.2 Run Configuration

Table 7 summarizes the configuration used for the reported runs.

Setting	Value
Dataset subset	Who&When Hand-Crafted
Number of traces	58
Model	`gpt-4.1-mini`
Temperature	0
Scope budget	5 selected reference steps
Random seed	42
Final attribution max output	300 tokens
EASD scope max output	800 tokens
Source-candidate-pool scope max output	1000 tokens
Source-candidate-pool max requirements	6
Source-candidate-pool max symptoms	5
Source-candidate-pool chunk size / overlap	25 / 5 steps
Source-candidate-pool max candidate pool	20 candidates
Candidate neighborhood before / after	1 / 2 steps

Table 7: Run configuration for the reported experiments.

A.3 Prompt Components

All prompts render the task and trace as data. Evaluation-only fields such as the gold responsible agent, gold decisive failure step, and gold reason are not included in the prompt text.

The final attribution prompt is implemented in `src/final_localizer.py`. It asks for exactly one responsible agent and one decisive step, returned as JSON. For scope-guided methods, the same prompt receives an additional reference block containing the selected steps. The reference steps guide attention, but the final judge is still allowed to choose any valid step from the full trace.

The generic LLM scope selector is implemented in `src/scope_methods.py`. It receives the full rendered trace and returns up to five selected step indices. It does not use specialized failure categories.

The EASD adapter is implemented in `src/easd_adapter.py`. It uses two selector calls: one for overstep-like behavior and one for loop-like behavior. The outputs are parsed separately, filtered to valid trace steps and expected item types, deduplicated,

and merged into one selected reference-step set. This is an adapter for this paper's trace format and not a full reproduction of the original scope-delineation system.

The source-candidate-pool selector is implemented in `src/source_candidate_pool.py`. It first builds a task brief and a failure-symptom summary. It then splits the trace into overlapping chunks and asks for source, commitment, failed-recovery, and premature-termination candidates inside each chunk. Candidate steps from all chunks are deduplicated into a bounded pool. A final consolidation call selects up to five source candidates from this pool, and their step indices are sorted before being passed to the final attribution judge.

A.4 Parsing and Validity Checks

All LLM outputs are parsed as JSON. A final attribution output is valid only if the predicted responsible agent matches one of the allowed agent names and the predicted decisive step is a valid step index from the trace. Invalid or unparsable final attribution outputs are counted as localization errors and as incorrect predictions.

Scope-selection outputs are normalized to valid trace indices where possible. For the generic selector, a parsing failure produces an empty selected scope and an error row. For the EASD adapter, overstep and loop outputs are parsed separately and then merged. For the source-candidate-pool selector, stage errors are recorded. If at least one selected step remains, the row can still be used for guided attribution. If no selected step remains, the scope row is marked as an error.