Scope Delineation Before Localization: A Two-Stage Framework for Enhancing Failure Attribution in Multi-Agent Systems

Kai Sun[1,2], **Wenqiang Li**[1,2], **Bo Dong**[2,3*], **Yuxin Lin**[1,2], **Jingyao Zhang**[4], **Bin Shi**[1,2]

[1]School of Computer Science and Technology, Xi'an Jiaotong University, China
[2]Shaanxi Provincial Key Laboratory of Big Data Knowledge Engineering, Xi'an Jiaotong University, China
[3]School of Distance Education, Xi'an Jiaotong University, China
[4]School of Advanced Technology, Xi'an Jiaotong-Liverpool University, China
sunkai@xjtu.edu.cn, liwenqiang@stu.xjtu.edu.cn, dong.bo@xjtu.edu.cn
yuxinlin@stu.xjtu.edu.cn, Jingyao.Zhang23@student.xjtlu.edu.cn, shibin@xjtu.edu.cn

Abstract

Large language models (LLMs) are seeing growing adoption in multi-agent systems. In these systems, efficient failure attribution is critical for ensuring robustness and interpretability. Current LLM-based attribution methods often face challenges with lengthy logs and lacking expert knowledge. Drawing inspiration from human debugging strategies, we propose an automated failure attribution framework, Scope Delineation Before Localization, which operates in two key stages: (1) identifying the failure scope and (2) pinpointing the failure step. By decoupling failure attribution into the two stages, our approach alleviates the reasoning workload of LLMs, enabling more precise failure attribution. To support scope delineation, we further introduce two strategies: Stepwise Scope Delineation and Expertise-Assisted Scope Delineation. Experiments on the Who&When dataset validate the efficacy of our two-stage framework, demonstrating substantial improvements over prior methods (up to 24.27% on step-level accuracy).

Code — https://github.com/Wen-qiangLi/SDBL

Introduction

With advanced capabilities in knowledge integration (Abu-Rasheed, Weber, and Fathi 2024; Feng, Zhang, and Fei 2023), instruction adherence (Murugadoss et al. 2025; Li et al. 2023b), and autonomous decision-making (Xu et al. 2025; Newsham and Prince 2025), large language models (LLMs) now serve as foundational components for multi-agent systems (MAS) (Li et al. 2024; Han et al. 2024). As MAS architectures grow more complex, failure attribution has become critical to ensuring system reliability (Triantafyllou, Singla, and Radanovic 2021; Zhang et al. 2025).

Failure attribution aims to identify the root causes of system failures in MAS. This process involves analyzing execution logs generated from MAS. Traditional approaches often rely on manual log reviews, which is time-consuming and requires specialized expertise (Zhuge et al. 2024; Jimenez et al. 2023). As MAS grow in scale and sophistication, these limitations become increasingly problematic. Consequently, automating failure attribution has emerged as a critical step to enhance system robustness (Zhang et al. 2025).

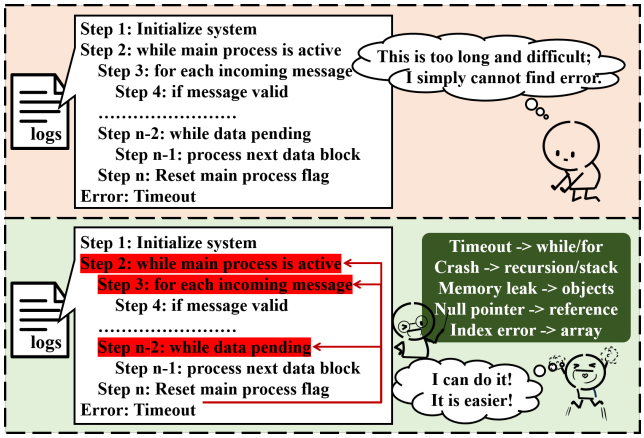

Figure 1: How do novice debuggers and experienced debuggers review code ?

To advance automatic failure attribution in MAS, Zhang et al. (2025) recently introduced the Who&When dataset, which is collected from 54 manually designed multi-agent systems and 126 algorithmically generated multi-agent systems. Each sample contains an execution log that records sequential agent actions. The task involves pinpointing the failure source (the earliest step leading to system failure) from the log.

Building on this benchmark, Zhang et al. (2025) proposed three LLM-based methods for this task: ALL_AT_ONCE, STEP_BY_STEP and BINARY_SEARCH. The ALL_AT_ONCE method analyzes the entire execution log in a single pass, whereas STEP_BY_STEP iteratively examines subsequences of the log until the failure source is identified. Differently, BINARY_SEARCH employs a recursive halving strategy, repeatedly narrowing the search space to isolate the failure source.

Despite leveraging state-of-the-art LLMs (e.g., OpenAI-o1 (Jaech et al. 2024), GPT-4o (Hurst et al. 2024), and

*Corresponding author

DeepSeek-R1 (Guo et al. 2025)) as backbones, experiments on the Who&When benchmark reveal persistent challenges for these methods. The highest agent accuracy (identifying the faulty agent) is only 53.44%, while step accuracy (identifying the exact failure step) stagnates at 16.59%. Our analysis reveals that two primary factors may cause this performance gap: **(1) Inherent task complexity.** The manually curated subset of Who&When features logs averaging 50 steps, with some exceeding 130 steps. Localizing failures in such lengthy logs, especially late-stage failures, is super difficult. **(2) Limited expertise.** Current state-of-the-art LLMs used for failure attribution are general-purpose models which lacks specialized knowledge of failure patterns in MAS. This limitation hinders their localization precision.

To address these challenges, we propose an automated failure attribution framework, Scope Delineation Before Localization (SDBL). The core innovation lies in decoupling failure attribution into: (1) delineating a failure-containing scope, followed by (2) identifying the precise failure step. As illustrated in Figure 1, SDBL mirrors human debugging strategies. Novice debuggers often review lengthy codes from the beginning due to limited expertise. The process is inefficient and time-consuming. In contrast, experienced debuggers narrow suspicious regions using execution outcomes or expertise. For example, timeout usually prompts inspection of loops (e.g., for or while cycles); Undefined variable errors trigger scrutiny of variable declaration scopes. Notably, scope delineation represents a simpler task for LLMs. By decoupling the single-pass analysis into a chain-of-thought process, this two-stage approach reduces the LLM's reasoning burden, enabling more accurate attribution.

To facilitate scope delineation, we introduce two strategies: Stepwise Scope Delineation and Expertise-Assisted Scope Delineation. The former employs an iterative step-by-step analysis to identify potential failure points using only the intrinsic knowledge of LLMs. The latter integrates predefined expertise paradigms, such as detecting unauthorized actions or command loops, to enhance the identification of failure-prone steps.

We validate our framework on the Who&When benchmark. Results show that Stepwise Scope Delineation boosts step accuracy from 8.77% to 25.93% for Hand-Crafted logs and 16.59% to 32.54% for Algorithm-Generated logs, showcasing its effectiveness. Integrating expert paradigms further improves accuracy to 27.78% (Hand-Crafted) and 33.33% (Algorithm-Generated), respectively, underscoring their significant impact on failure attribution performance for general LLMs.

Our contributions are summarized as follows:

- We introduce a two-stage Scope Delineation Before Localization framework for enhancing failure attribution in the multi-agent system.
- We present two strategies: Stepwise Scope Delineation and Expertise-Assisted Scope Delineation, which significantly enhance the attribution precision.
- Comprehensive experiments on the Who&When benchmark demonstrate notable gains in step accuracy, particularly for lengthy and intricate logs.

Related Work

LLM-based Multi-Agent System

Recent advances in LLMs enhance knowledge integration, instruction following, and diverse task handling via tool use (Deng et al. 2023; Xie et al. 2024), enabling sophisticated intelligent agents (Wang et al. 2024; Achiam et al. 2023). While single LLM agents excel at simple tasks (Yao et al. 2023; Zhang et al. 2023), they struggle in complex scenarios requiring collaboration and task decomposition. Consequently, LLM-based Multi-Agent Systems (Hong et al. 2023; Li et al. 2023a) leverage agent interaction and information sharing to improve performance in tasks like math reasoning (Madaan et al. 2023; Paul et al. 2023) and dialogue (He, Treude, and Lo 2025). Despite this progress, failure attribution within MAS remains underexplored.

LLM for Judging

Recent work explores using LLMs as automated evaluators (Gu et al. 2024; Hu et al. 2024; Li et al. 2023c). For example, Chan et al. (2023) applied them to assess open-ended question answers, lowering manual annotation costs. In mathematical reasoning, Madaan et al. (2023) used LLMs to provide feedback on intermediate results for refinement. Paul et al. (2023) developed REFINER, fine-tuning LLMs to generate explicit reasoning steps and then employing a critic model for feedback. Significantly, Zhang et al. (2025) pioneered LLMs for MAS failure attribution, pinpointing responsible agents and error steps, and proposed three automated methods.

Credit Allocation

In MAS, agents often affect each other. Simple global rewards or treating agents as independent cannot fully capture individual roles. An effective evaluation mechanism is needed to reflect each agent's value precisely. The goal of credit allocation is to measure each agent's or decision step's real contribution to a task (Li, Zou, and Liu 2025; Qian et al. 2025; Wang et al. 2025). However, the focus of failure attribution and credit allocation is different: failure attribution aims to find the failure source of a system, while credit assignment is to measure the contribution of every agent, no matter the outcome. Credit allocation may be guided by insights from failure attribution, suggesting that a downstream perspective could be particularly informative.

Methodology

We begin by formally describing the failure attribution problem. Next, we present the preliminaries by reviewing three established baselines for this task. We then introduce the Scope Delineation Before Localization (SDBL) framework, and present two strategies for scope delineation: Stepwise Scope Delineation and Expertise-Assisted Scope Delineation. The overview of our framework is presented in Figure 2.

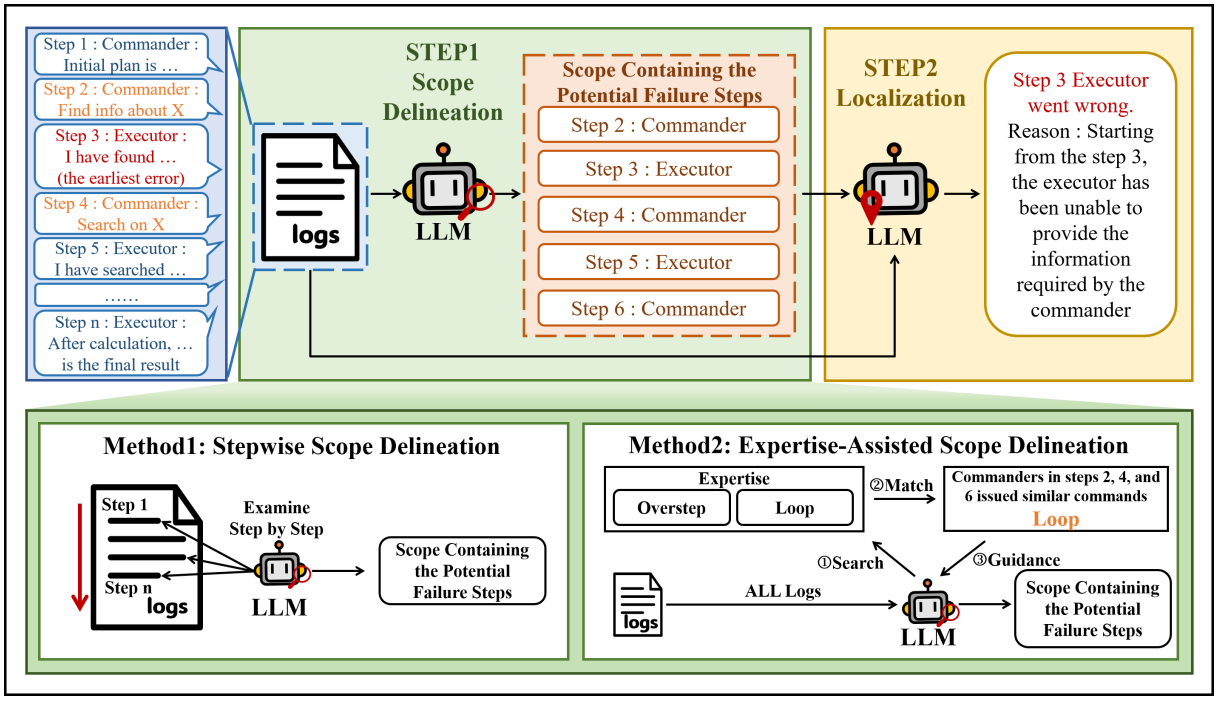

Figure 2: Overview of our Scope Definition Before Localization (SDBL) framework.

Problem Definition

Given an execution log $L = \{s_1, s_2, \ldots, s_n\}$ containing n steps, where each step $s_t = (t, r_t, c_t)$ records the step index t, the content c_t produced by agent r_t, with $r_t \in \mathcal{A}$ representing the agent's role (e.g., Orchestrator, WebSurfer) and $\mathcal{A}$ representing the set of roles, the objective of failure attribution is identify the agent and earliest step responsible for the system failure. Formally, the failure attribution process is defined as:

$$(r_t, t) = f(L) \tag{1}$$

where $f(\cdot)$ denotes the failure attribution function that maps the log L to a tuple containing the agent role r_t and the step index t.

Preliminaries

ALL_AT_ONCE The ALL_AT_ONCE method processes the entire execution log L through a LLM $\mathcal{M}$. This approach analyzes the full sequence in a single pass to identify the failure source:

$$(\hat{r}_t, \hat{t}) = f_{\text{ALL\_AT\_ONCE}}(L) \tag{2}$$

where $f_{\text{ALL\_AT\_ONCE}}(\cdot)$ denotes the full-sequence analysis function modeled by $\mathcal{M}$.

STEP_BY_STEP The STEP_BY_STEP method iteratively analyzes log subsequences $L_{\{1:1\}}$, $L_{\{1:2\}}$,...,$L_{\{1:k\}}$ until the failure source is identified, where $L_{\{1:k\}} = \{s_1, ..., s_k\}$ $(1 \le k \le n)$. At each iteration k, $\mathcal{M}$ determines whether step s_k in $L_{\{1:k\}}$ is the failure source, producing response $O \in \{\text{Yes}, \text{No}\}$. Formally, this iterative process is defined as:

$$f_{\text{STEP\_BY\_STEP}}(L_{\{1:k\}}) = \begin{cases} \text{Terminate and return } (\hat{r}_k, \hat{k}), & \text{if } O = \text{Yes} \\ f_{\text{STEP\_BY\_STEP}}(L_{\{1:k+1\}}), & \text{if } O = \text{No} \\ \text{Terminate}, & \text{if } k > n \end{cases} \tag{3}$$

where $f_{\text{STEP\_BY\_STEP}}(\cdot)$ denotes the stepwise analysis function modeled by $\mathcal{M}$.

BINARY_SEARCH The BINARY_SEARCH method employs a recursive halving strategy. The $\mathcal{M}$ analyzes successive segments $L_{\{j:k\}} = \{s_j, ..., s_k\}$, bisecting the search space by examining the left half $L_{\{j:m\}}$ or the right half $L_{\{m+1:k\}}$ (where $m = \lfloor (j+k)/2 \rfloor$). The recursion terminates when the segment reduces to a single step:

$$f_{\text{BINARY\_SEARCH}}(L_{\{j:k\}}) = \begin{cases} \text{Terminate and return } (\hat{r}_j, \hat{j}), & \text{if } j = k \\ f_{\text{BINARY\_SEARCH}}(L_{\{j:m\}}), & \text{if failure in left} \\ f_{\text{BINARY\_SEARCH}}(L_{\{m+1:k\}}), & \text{if failure in right} \end{cases} \tag{4}$$

where the segment $L_{\{j:k\}}$ is initialized with $j = 1, k = n$, and $f_{\text{BINARY\_SEARCH}}(\cdot)$ denotes the segmentation function modeled by $\mathcal{M}$.

The ALL_AT_ONCE method requires LLMs to identify failure points from lengthy or intricate logs during single-pass analysis. The STEP_BY_STEP method struggles with restricted follow-up context during incremental evaluation, whereas BINARY_SEARCH is hindered by missing prior context when narrowing down via bisection. Additionally, general-purpose LLMs (e.g., DeepSeek, GPT-4)

often lack expertise for precise failure attribution. These factors heighten the difficulties of accurate failure attribution in multi-agent systems.

Scope Delineation Before Localization

To address these issues, we present Scope Delineation Before Localization (SDBL), a two-stage framework that first defines the scope of the potential failure source and then localizes the critical failure step. The key innovation of our method—and its distinction from prior works—lies in defining the failure-containing scope. To achieve this, we introduce two strategies: Stepwise Scope Delineation and Expertise-Assisted Scope Delineation.

During scope delineation, we insert the potential failure steps into a scope set S, with hyperparameter N as its maximum size.

Stepwise Scope Delineation This strategy adapts the STEP_BY_STEP method to iteratively expand S. When a failure step $(\hat{r}_t, \hat{t})$ is detected, the process continues rather than terminating: We add $(\hat{r}_t, \hat{t})$ to S and persist until $|S| = N$. Formally, this process is as follows:

$$f_{\text{SSD}}(L_{\{1:k\}}) = \begin{cases} (\hat{r}_k, \hat{k}) \to S, \text{then } f_{\text{SSD}}(L_{\{1:k+1\}}) & \text{if } O = \text{Yes } \& \ |S| \neq N \\ f_{\text{SSD}}(L_{\{1:k+1\}}), & \text{if } O = \text{No} \\ \text{Terminate} & \text{if } |S| = N \mid k > n \end{cases} \tag{5}$$

where the subsequence $L_{\{1:k\}}$ starts with $k = 1$. The function $f_{\text{SSD}}(\cdot)$ determines whether the step k contains a failure step, returning a binary response $O \in \{\text{Yes, No}\}$.

We adopt STEP_BY_STEP based on two principles. First, empirical results have demonstrated STEP_BY_STEP's superior capability on identifying failure steps (Zhang et al. 2025). Second, by relying solely on LLMs' intrinsic knowledge during scope definition, we can isolate the framework's inherent effectiveness while minimizing external expertise integration.

Expertise-Assisted Scope Delineation Alternatively, this strategy leverages predefined expertise to refine failure scope delineation. Two paradigms are introduced: Overstep and Loop. These paradigms guide the identification of failure-prone steps, as described below.

> **Overstep: Detecting Unauthorized Agent Actions.**
> *An agent overstep when it issues or executes commands beyond its designated authority.*

To operationalize this paradigm, we first predefine the role-based agent authorities and encode them into prompts. The LLM is then tasked with detecting steps where agents exceed these boundaries. For example, commanders are responsible for scheduling and terminating processes while executors are responsible for executing commands and reporting outcomes. By design, termination commands are restricted to commanders. If an executor issues termination, this constitutes overstepping. The corresponding steps are added to scope set S.

> **Loop: Identifying Repeated Commands.**
> *A loop occurs when an agent (e.g., a commander) repeatedly issues similar commands.*

In normal turns, commanders await executor responses before proceeding. Repeated commands often signal execution failures or unacknowledged results, risking infinite loops. For example, if a commander re-sends the same (or similar) command, we flag the initial command and its subsequent executor responses as potential failures. All related steps are added to scope set S.

While additional paradigms could be incorporated to enhance performance, we focus here on the two simple yet frequent in MAS failure. Formally, we obtain the scope set S by applying these two paradigms sequentially:

$$S = f_{\text{overstep}}(L) \cup f_{\text{loop}}(L) \tag{6}$$

where $f_{\text{overstep}}(\cdot)$ and $f_{\text{loop}}(\cdot)$ denote scope delineation functions guided by the Overstep and Loop paradigms, respectively. Both functions identify failure-prone steps from the log L, and their union forms S. If the total number of steps exceeds N, truncation is applied.

Localization To this end, we leverage the set scope S to refine failure localization. Specifically, we augment the log L with critical insights from S through targeted prompting. This focuses the model's analysis on high-risk regions. During localization, the LLM receives the explicit instruction: "*The following agents and steps are flagged for special attention (from reference_content):* $\{S\}$." Formally, the process of localization is as follows:

$$(\hat{r}_t, \hat{t}) = f_{\text{LOC}}(L, S) \tag{7}$$

where $f_{\text{LOC}}(\cdot)$ denotes the localization function that predicts the failure source $(\hat{r}_t, \hat{t})$ by synthesizing evidence from both L and S.

Experiment

Setup

Dataset We evaluate our method on the Who&When dataset released by (Zhang et al. 2025). This dataset explicitly annotates both failure step ("When") and responsible agents ("Who"). Due to the intensive labeling costs, the dataset only contains a test set comprising two distinct subsets: the Hand-Crafted subset and the Algorithm-Generated subset. Each sample in the subset includes: (1) execution logs with agent identifiers and outputs, and (2) annotated failure steps, responsible agents, and failure reasons. Detailed dataset statistics are presented in Table 1.

Baselines Given the novel nature of this task, we compare our approach against three established failure attribution baselines from (Zhang et al. 2025): ALL_AT_ONCE, STEP_BY_STEP and BINARY_SEARCH. To assess the efficacy of our scope delineation methods, we also evaluate our framework's performance across three strategies: Stepwise Scope Delineation (SSD), Expertise-Assisted Scope

Dataset	#Num	#Avg.Steps	#Avg.Roles
HC	54	50.00	6.59
AG	126	8.72	3.63

Table 1: Statistics of the Who&When dataset which comprises two subsets: Hand-Crafted (HC) and Algorithm-Generated (AG). Total number of logs per subset (#Num), average steps per log (#Avg.Steps), and average distinct roles per log (#Avg.Roles) are presented for HC and AG, respectively.

Delineation (EASD) and Random Scope Delineation (RSD). The RSD strategy involves constructing the scope set by randomly sampling steps from the execution log.

LLM Backbones Following Zhang et al. (2025)'s experimental setups, we employ GPT-4o as the primary LLM backbone for fair comparison. To assess the generalization of our method, we extend evaluation to additional state-of-the-art models including DeepSeek-R1 and DeepSeek-V3.

Evaluation Protocol Following prior work (Zhang et al. 2025), we employ two evaluation metrics: agent accuracy and step accuracy. Agent accuracy measures whether methods correctly identify the responsible failure agent, while step accuracy measures whether methods accurately identify the exact failure step.

To examine performance on scope delineation, we additionally introduce the Hit@K metric, where K represents the size of the scope set S generated by the scope delineation method. Given m execution logs, let t_i denote the ground-truth failure step, r_i the ground-truth role of failure agent, and S_i the delineated scope for the i-th log. Hit@K is calculated as:

$$\text{Hit@K} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\{(r_i, t_i) \in S_i\} \tag{8}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function. This metric quantifies how frequently the true failure step resides within the delineated scope.

Implementation Details For each execution log, agents are tasked with addressing human-posed queries. While the ground truth answer to each query is provided, real-world failure attribution often lacks access to such answers. Thus, we evaluate all methods without using the information of answers. We introduce a hyperparameter N to limit the maximum size of the scope set S. For Hand Crafted subset, N is chosen from [3, 5, 7, 9] with $N = 5$ being the best configuration. In the Algorithm Generated subset, where execution logs average 8 steps, we set $N = 3$. To minimize fluctuations in LLMs output, we set the temperature to 0.

Main Results

In this section, we compare our method against prior baselines, as presented in Table 2.

The results demonstrate that SDBL (EASD) consistently outperforms previous methods across both datasets. Notably,

Method	HC		AG	
	AGENT	STEP	AGENT	STEP
ALL_AT_ONCE	53.44	3.51	51.12	13.53
STEP_BY_STEP	32.75	8.77	26.02	15.31
BINARY_SEARCH	36.21	6.90	30.11	16.59
SDBL (RSD)	62.96	14.81	56.35	31.75
SDBL (SSD)	61.11	25.93	64.29	32.54
SDBL (EASD)	**62.96**	**27.78**	**68.25**	**33.33**
SDBL (EASD)	**62.96**	**27.78**	**68.25**	**33.33**

Table 2: Comparison of agent accuracy (AGENT) and step accuracy (STEP) across Hand-Crafted (HC) and Algorithm-Generated (AG) subsets.

even with randomized scope definitions, SDBL (RSD) maintains performance advantages over the ALL_AT_ONCE, STEP_BY_STEP and BINARY_SEARCH baselines. On the Algorithm-Generated (AG) dataset, we observe particularly significant improvements in step accuracy. For instance, SDBL (RSD) achieves an 18.22% improvement over the ALL_AT_ONCE method. The condensed nature of AG logs (shorter length) likely contributes to this performance, where the failure steps are easily covered by the delineated scope. We find that all three SDBL variants show similar step accuracy (i.e., around 30%).

In contrast, the Hand-Crafted (HC) sub-dataset presents greater challenges due to its lengthy execution logs, as the statistics shown in Table 1. This results in significantly lower step accuracy for prior baselines (e.g., the ALL_AT_ONCE method achieves only 3.51% step accuracy). Our framework exhibits strong improvements on this dataset. Specifically, SDBL (SSD) outperforms its RSD counterpart by 11.12% on step accuracy. Furthermore, the SDBL (EASD) method achieves the best performance on both accuracy metrics, revealing that even powerful LLM backbone (i.e., GPT-4o) benefits from predefined expertise when tackling failure attribution.

Overall, these findings confirm the effectiveness of our two-stage architecture, demonstrating that both scope delineation strategies enhance the identification of failure steps.

Performance on Different LLMs

In this section, we examine the performance of our method under different LLMs, as summarized in Table 3. The results demonstrate that even when equipped with advanced LLMs, incorporating predefined expertise remains beneficial for this task. Specifically, SDBL (EASD) achieves consistent performance gains over prior baselines on both datasets when utilizing the GPT-4o and DeepSeek-V3 models. On DeepSeek-R1, SDBL (EASD) exhibits superior step accuracy and competitive agent accuracy. SDBL (SSD)'s superior performances over SDBL (RSD) and prior baselines further prove the effectiveness of the two-stage framework under different LLMs. These results underscore the generalization of our method, highlighting its robustness.

Further Analysis

We conduct a further analysis to examine the behavior of our proposed methods. Due to space limitations, we focus this

	Method	GPT-4o		DeepSeek-R1		DeepSeek-V3	
		AGENT ACC	STEP ACC	AGENT ACC	STEP ACC	AGENT ACC	STEP ACC
Algorithm Generated	ALL_AT_ONCE	51.12	13.53	53.17	19.84	36.51	19.84
	STEP_BY_STEP	26.02	15.31	57.14	25.40	22.22	3.97
	BINARY_SEARCH	30.11	16.59	57.14	30.95	57.14	10.32
	SDBL (RSD)	56.35	31.75	57.94	34.13	55.56	27.78
	SDBL (SSD)	64.29	32.54	**64.29**	35.71	63.49	30.95
	SDBL (EASD)	**68.25**	**33.33**	56.35	**35.71**	**63.49**	**39.68**
Hand Crafted	ALL_AT_ONCE	53.44	3.51	55.56	1.85	62.96	1.85
	STEP_BY_STEP	32.75	8.77	50.00	9.26	38.89	7.41
	BINARY_SEARCH	36.21	6.90	**55.56**	9.26	59.26	12.96
	SDBL (RSD)	62.96	14.81	50.00	11.11	59.26	16.67
	SDBL (SSD)	61.11	25.93	50.00	12.96	59.26	25.93
	SDBL (EASD)	**62.96**	**27.78**	50.00	**20.37**	**68.52**	**27.78**

Table 3: Performance comparison under different LLMs.

Method	Hit@3	Hit@5	Hit@7	Hit@9
ALL_AT_ONCE	12.07	19.83	30.17	37.07
STEP_BY_STEP	14.66	16.38	18.10	31.90
ALL_AT_ONCE*	9.26	14.81	25.93	25.93
SDBL (SSD)	20.37	35.19	35.19	35.19
SDBL (EASD)	**25.93**	**42.59**	**42.59**	**55.56**

Table 4: Performance comparison on the Hit@K metric, with $K = 3, 5, 7, 9$.

	AGENT	STEP	TOKEN
ALL_AT_ONCE	53.44	3.51	17106
STEP_BY_STEP	32.75	8.77	87720
BINARY_SEARCH	36.21	6.90	34659
SDBL (SSD)	61.11	25.93	111310
SDBL (EASD)	**62.96**	**27.78**	24657

Table 5: Agent accuracy (AGENT) and step accuracy (STEP), alongside token consumption (TOKEN), across different methods.

investigation on the Hand-Crafted subset, where longer and more complex log structures facilitate a rigorous evaluation of different methods. GPT-4o serves as the primary model.

Comparison on Scope Delineation This section examines the performance of our method on scope delineation. Specifically, we compare our methods, SDBL (SSD) and SDBL (EASD), against prior baselines, ALL_AT_ONCE, STEP_BY_STEP and the variant ALL_AT_ONCE*, on the Hit@K metric ($K = 3, 5, 7, 9$). Both the ALL_AT_ONCE and STEP_BY_STEP methods predict a single failure step and expands it into a window of K steps centered around the predicted step, with a tolerance (e.g., ± 2 for $K = 5$). In contrast, ALL_AT_ONCE* directly prompts the LLM to predict K failure-prone steps, which are not necessarily continuous.

As K increases, all methods exhibit consistent improvements, which is expected. Notably, STEP_BY_STEP outperforms ALL_AT_ONCE at $K = 3$, but this trend reverses as K grows larger, suggesting the importance of utilizing more context to identify potential failure steps. SDBL (SSD), which actually incorporates more context than STEP_BY_STEP at each K, demonstrates superior performance over ALL_AT_ONCE at $K = 3, 5, 7$ and achieves competitive results at $K = 9$. Meanwhile, ALL_AT_ONCE* consistently underperforms.

These results show the superiority of Stepwise Scope Delineation (SSD) in identifying the failure-containing scope. Furthermore, SDBL (EASD) achieves the highest performance across all K values, further demonstrating the value of integrating expertise into scope delineation.

Comparison on Token Consumption This section compares the token consumption for different failure attribution methods, as illustrated in Table 5. The ALL_AT_ONCE method demonstrates the lowest token usage due to its single-pass analysis of the execution log. In contrast, the STEP_BY_STEP method incurs the high token consumption, as it iteratively analyzes execution logs for multiple times. Notably, when the failure is predicted near the end of the log, this approach leads to high token usage. The BINARY_SEARCH method achieves greater efficiency by isolating relevant log segments in each iteration, halving the search space.

For the SDBL (SSD) method, the Stepwise Scope Delineation builds on STEP_BY_STEP to achieve good performances, albeit at the cost of increased token consumption. Conversely, SDBL (EASD) pinpoint failure-prone steps in a single pass, achieving robust balance between accuracy and cost, with the token consumption marginally higher than ALL_AT_ONCE.

Performance on Different Log Lengths We evaluate agent and step accuracy across different log lengths, as illustrated in Figure 3. Test samples are grouped into six tiers based on log step counts: Level 1 (5–16 steps), Level 2 (17–21), Level 3 (24–32), Level 4 (33–53), Level 5 (59–106), and Level 6 (113–130).

For agent accuracy, performance variations among methods persist across all log length levels. Step accuracy, however, reveals stark contrasts: both ALL_AT_ONCE and STEP_BY_STEP exhibit declines, nearing zero once logs exceed 24 steps (Level 3), highlighting limitations of these

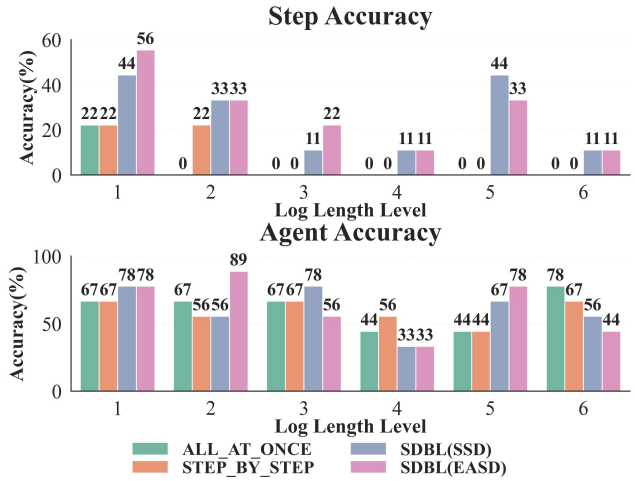

Figure 3: Bar charts presenting agent accuracy and step accuracy under different levels of log length.

	AGENT ACC	STEP ACC
SDBL (EASD)	**62.96**	**27.78**
w/o Overstep	59.26	18.52
w/o Loop	57.41	12.96
w/o All	53.44	3.51

Table 6: Ablation study assessing the contribution of two paradigms in Expertise-Assisted Scope Delineation.

baselines in processing lengthy logs. In contrast, our SDBL (SSD) and SDBL (EASD) methods sustain superior performance, particularly on lengthy logs. This demonstrates the superiority of our two-stage framework for failure attribution in long, intricate multi-agent interactions.

Ablation Study on Expertise Paradigms We conduct ablation studies to evaluate the contributions of the Overstep and Loop paradigms in the Expertise-Assisted Scope Delineation framework. Agent accuracy and step accuracy for three ablated methods are presented in Table 6.

The results demonstrate that disabling either paradigm significantly diminishes both accuracy metrics, underscoring their necessity for effective failure attribution guidance. Notably, removing the Loop paradigm causes a more pronounced performance degradation, emphasizing its critical role in failure localization. Practical scenarios often involve logs with complex loops and repetitive sequences. Without the Loop paradigm, we found that LLMs struggled to isolate the failure source (i.e., the earliest failure step).

Case study In this section, we select two examples from the Hand Crafted dataset and present visualizations of predictions by SDBL (EASD) and ALL_AT_ONCE in Figure 4.

In Case1, the question posed by human was: "What was the volume in m^3 of the fish bag calculated in the University of Leicester paper 'Can Hiccup Supply Enough Fish to Maintain a Dragon's Diet?"' The ground-truth failure source is labeled at step 4, where the WebSurfer agent extracted incomplete data from the PDF. The ALL_AT_ONCE method

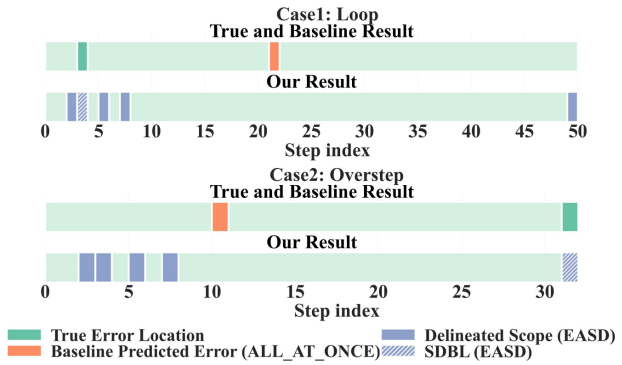

Figure 4: Case study on two samples selected from the Hand Crafted dataset.

incorrectly pinpointed step 22 (a subsequent commander warning) as the failure source.

Guided by the Loop paradigm, our method first defines a scope set S comprising high-risk steps (i.e., steps 3, 4, 6, 8, and 50). Upon closer examination, we observed that steps 3 and 6 contain duplicate commander instructions, while steps 4 and 8 represent the WebSurfer's responses. By constraining the model's analysis to these critical steps via the scope set, SDBL (EASD) accurately localized the failure to step 4.

In Case2, the question posed by human was: "In the endnote found in the second-to-last paragraph of page 11 of the book with doi 10.2307/j.ctv9b2xdv, what date in November was the Wikipedia article accessed?" The ground-truth failure source is labeled at step 32, where FileSurfer terminated itself without reporting to the Orchestrator. The ALL_AT_ONCE method incorrectly flagged step 11.

Our method identified a scope set S (i.e., steps 3, 4, 6, 8 and 32). While steps 3, 4, 6, and 8 were included in S (induced by the Loop paradigm) due to command similarities, we observed that these commands were progressively distinct rather than identical. The LLM is clever to pinpoint the highest-risk step 32 among these steps. The step 32 is induced by the Overstep paradigm, where FileSurfer issued a termination command beyond its authority.

Conclusion

In MAS failure attribution, existing methods struggle with locating failures in lengthy execution logs and often lack the expertise for precise attribution. To overcome these limitations, we propose Scope Delineation Before Localization (SDBL), which identifies potential failure scopes before pinpointing critical steps. This two-stage structure decouples the analysis, reducing the reasoning burden on LLMs and improving attribution accuracy. We further enhance SDBL with Stepwise and Expertise-Assisted Scope Delineation strategies, both substantially boosting performance. Evaluations on the Who&When dataset demonstrate SDBL's significant improvements in agent- and step-level accuracy, especially for long or complex logs.

Acknowledgments

This research was partially supported by the Key Research and Development Project in Shaanxi Province No. 2023GXLH-024, the National Natural Science Foundation of China No. 62406242, 62476215, 62302380, 62037001, 62137002 and 62192781, Project of China Knowledge Centre for Engineering Science and Technology.

References

Abu-Rasheed, H.; Weber, C.; and Fathi, M. 2024. Knowledge graphs as context sources for llm-based explanations of learning recommendations. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, 1–5. IEEE.

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Chan, C.-M.; Chen, W.; Su, Y.; Yu, J.; Xue, W.; Zhang, S.; Fu, J.; and Liu, Z. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36: 28091–28114.

Feng, C.; Zhang, X.; and Fei, Z. 2023. Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs. *arXiv preprint arXiv:2309.03118*.

Gu, J.; Jiang, X.; Shi, Z.; Tan, H.; Zhai, X.; Xu, C.; Li, W.; Shen, Y.; Ma, S.; Liu, H.; et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Han, S.; Zhang, Q.; Yao, Y.; Jin, W.; and Xu, Z. 2024. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*.

He, J.; Treude, C.; and Lo, D. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5): 1–30.

Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S. K. S.; Lin, Z.; et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Hu, Z.; Zhang, J.; Xiong, Z.; Ratner, A.; Xiong, H.; and Krishna, R. 2024. Language model preference evaluation with multiple weak evaluators. *arXiv preprint arXiv:2410.12869*.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jaech, A.; Kalai, A.; Lerer, A.; Richardson, A.; El-Kishky, A.; Low, A.; Helyar, A.; Madry, A.; Beutel, A.; Carney, A.; et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Li, G.; Hammoud, H.; Itani, H.; Khizbullin, D.; and Ghanem, B. 2023a. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36: 51991–52008.

Li, M.; Zhang, Y.; Li, Z.; Chen, J.; Chen, L.; Cheng, N.; Wang, J.; Zhou, T.; and Xiao, J. 2023b. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. *arXiv preprint arXiv:2308.12032*.

Li, X.; Wang, S.; Zeng, S.; Wu, Y.; and Yang, Y. 2024. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1): 9.

Li, X.; Zhang, T.; Dubois, Y.; Taori, R.; Gulrajani, I.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023c. Alpacaeval: An automatic evaluator of instruction-following models.

Li, X.; Zou, H.; and Liu, P. 2025. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36: 46534–46594.

Murugadoss, B.; Poelitz, C.; Drosos, I.; Le, V.; McKenna, N.; Negreanu, C. S.; Parnin, C.; and Sarkar, A. 2025. Evaluating the evaluator: Measuring llms' adherence to task evaluation instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 19589–19597.

Newsham, L.; and Prince, D. 2025. Personality-Driven Decision-Making in LLM-Based Autonomous Agents. *arXiv preprint arXiv:2504.00727*.

Paul, D.; Ismayilzada, M.; Peyrard, M.; Borges, B.; Bosselut, A.; West, R.; and Faltings, B. 2023. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*.

Qian, C.; Acikgoz, E. C.; He, Q.; Wang, H.; Chen, X.; Hakkani-Tür, D.; Tur, G.; and Ji, H. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Triantafyllou, S.; Singla, A.; and Radanovic, G. 2021. On blame attribution for accountable multi-agent sequential decision making. *Advances in Neural Information Processing Systems*, 34: 15774–15786.

Wang, F.; Zhang, Z.; Zhang, X.; Wu, Z.; Mo, T.; Lu, Q.; Wang, W.; Li, R.; Xu, J.; Tang, X.; et al. 2024. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*.

Wang, H.; Qian, C.; Zhong, W.; Chen, X.; Qiu, J.; Huang, S.; Jin, B.; Wang, M.; Wong, K.-F.; and Ji, H. 2025. Otc: Optimal tool calls via reinforcement learning. *arXiv e-prints*, arXiv–2504.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Hua, T. J.; Cheng, Z.; Shin, D.; Lei, F.; et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37: 52040–52094.

Xu, R.; Li, X.; Chen, S.; and Xu, W. 2025. Nuclear deployed: Analyzing catastrophic risks in decision-making of autonomous llm agents. *arXiv preprint arXiv:2502.11355*.

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

Zhang, J.; Krishna, R.; Awadallah, A. H.; and Wang, C. 2023. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*.

Zhang, S.; Yin, M.; Zhang, J.; Liu, J.; Han, Z.; Zhang, J.; Li, B.; Wang, C.; Wang, H.; Chen, Y.; et al. 2025. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*.

Zhuge, M.; Zhao, C.; Ashley, D.; Wang, W.; Khizbullin, D.; Xiong, Y.; Liu, Z.; Chang, E.; Krishnamoorthi, R.; Tian, Y.; et al. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*.