

Error-Propagation Modeling for Failure Attribution in LLM-Based Multi-Agent Systems

Jiaqi Liao¹, Yuanzhao Zhai¹, Huanxi Liu¹, Xu Zhang¹,
Zheming Zhuang², Dawei Feng^{1*}, Bo Ding³, Huaimin Wang³

¹College of Computer Science and Technology, National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment, Changsha, Hunan, China

²School of Mechanical Engineering, Tianjin University, Tianjin, China

³College of Computer Science and Technology, National University of Defense Technology; National Key Laboratory of Parallel and Distributed Computing, Changsha, Hunan, China

Abstract

LLM-based multi-agent systems (MASs) are increasingly used to solve complex tasks through coordinated reasoning, tool use, and interaction with external resources. However, attributing failures in such systems remains challenging because the observed outcome often does not directly reveal the error responsible for the failed execution. In this work, the attribution target is the decisive error, defined as the agent-step pair whose correction would recover the failed execution. Existing approaches largely identify suspicious steps without explicitly modeling how errors propagate across interactions or persist in unresolved loops, making decisive errors difficult to distinguish from downstream failure symptoms. We propose Error-Propagation Modeling for Failure Attribution (EMFA). EMFA constructs a structured representation of the failed trajectory, models both cascading propagation and persistent interaction loops, and uses propagation-aware candidate screening followed by counterfactual verification to identify the decisive agent-step pair. On the Who&When benchmark, EMFA achieves state-of-the-art step-level attribution accuracy and remains competitive at the agent level. It improves the previous best step-level results by 3.45 and 4.40 percentage points on the Hand-Crafted and Algorithm-Generated subsets, respectively.

Introduction

Multi-agent systems (MASs) have emerged as a promising paradigm for solving increasingly complex tasks involving tool use, reasoning, and agent coordination by orchestrating multiple LLM-based agents to enable task decomposition, collaborative reasoning, and interaction with external resources. However, the integration of heterogeneous agents makes it difficult to trace unsuccessful executions back to the responsible agent and step. Recent empirical studies show that even advanced MAS frameworks, such as HyperAgent (Phan et al. 2025), still experience substantial failure rates (74.7%) on the SWE-Bench Lite benchmark, with failures arising from various stages of multi-agent execution (Cemri et al. 2025). These observations motivate the problem of **failure attribution**. Specifically, under the Who&When formulation adopted in this work, the attribution target is the **decisive error**, defined as the agent-step pair whose correction would recover the failed execution (Zhang et al. 2025d). Identifying this decisive error provides actionable signals for

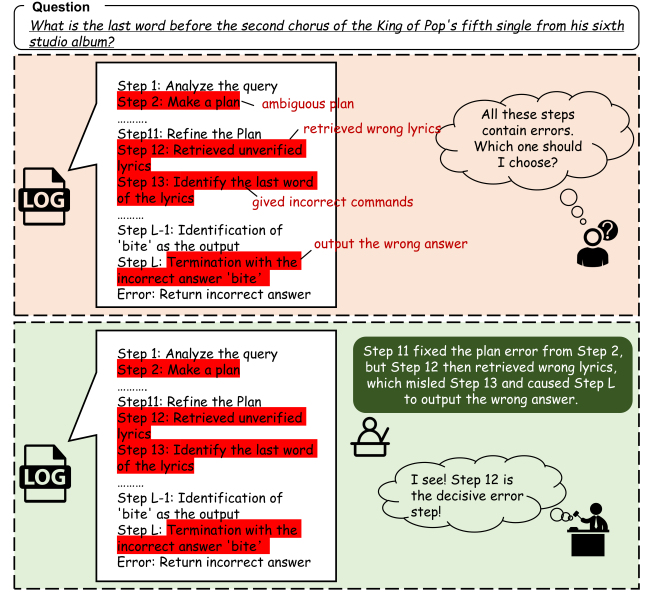


Figure 1: Illustration of why error-propagation modeling is needed to separate decisive errors from misleading failure symptoms.

debugging, targeted repair, and learning from failed executions. Nevertheless, reliable attribution remains challenging because failures are rarely isolated events: errors may either propagate through dependent interactions or remain unresolved in repetitive interaction loops, eventually manifesting as misleading failure symptoms.

Existing failure attribution approaches can be broadly categorized into four paradigms: direct LLM-based methods that infer attribution targets from textual trajectories (Zhang et al. 2025d; Banerjee, Nair, and Borogovac 2025; Sun et al. 2026); replay-based methods that estimate step responsibility through repeated trajectory executions (Ge et al. 2025); fine-tuning-based methods that learn specialized attribution models from additional supervision (Zhang et al. 2025a,b); and structure-guided methods that exploit dependency structures to identify candidate steps (Wang et al. 2026; Rafi et al. 2026). Despite their differences, these approaches largely treat attri-

*Co-corresponding author.

bution as selecting suspicious steps from an observed failure, without explicitly modeling how errors propagate across the trajectory or persist in unresolved interaction loops. As illustrated in Figure 1, a failed execution may contain multiple apparently erroneous steps, including deviations that are later corrected and downstream symptoms that merely inherit an earlier error. Identifying the decisive error therefore requires tracing how errors influence subsequent execution rather than judging individual steps in isolation. The figure illustrates cascading propagation across dependent steps, while failed trajectories may also contain persistent interaction loops in which agents repeatedly act without making task progress. Accordingly, we focus on these two complementary failure patterns.

Building on these observations, we propose **Error-Propagation Modeling for Failure Attribution (EMFA)**, a failure attribution framework that explicitly models how errors evolve within failed multi-agent executions. EMFA first transforms flat execution logs into structured dependency representations to recover how information flows across agents and steps. It then models two complementary failure patterns: cascading error propagation, where an early deviation affects subsequent decisions through dependencies, and persistent interaction loops, where agents repeatedly act without making substantive progress. Based on these structures, EMFA constructs candidate decisive errors and applies counterfactual verification to distinguish truly recoverable errors from downstream carriers and failure symptoms.

On the Who&When benchmark (Zhang et al. 2025d), EMFA achieves state-of-the-art step-level failure attribution accuracy, improving over the previous best by 3.45 and 4.40 percentage points on the Hand-Crafted and Algorithm-Generated subsets, respectively. It also achieves the best agent-level accuracy on the Hand-Crafted subset while remaining competitive on the Algorithm-Generated subset.

Our contributions are summarized as follows:

- We introduce an error-propagation-oriented perspective on failure attribution in LLM-based multi-agent systems. This perspective explicitly models how errors propagate across dependent steps or persist in unresolved interaction loops, enabling decisive errors to be distinguished from downstream failure symptoms.
- We propose **EMFA**, a failure attribution framework that organizes failed trajectories into structured dependency representations, models cascading error propagation and persistent interaction loops to screen candidate steps, and applies counterfactual verification to identify the decisive agent-step pair.
- Experiments on the Who&When benchmark show that **EMFA** achieves state-of-the-art step-level failure attribution accuracy while remaining competitive at the agent level.

Related Work

LLM-Based Multi-Agent Systems

Recent LLM-based multi-agent systems coordinate specialized agents through either manually specified workflows or

automatically optimized interaction structures. Representative workflow-based systems include AutoGen (Wu et al. 2023), MetaGPT (Hong et al. 2024), and ChatDev (Qian et al. 2024), while AFlow (Zhang et al. 2025c) optimizes agent workflows or structures automatically. Despite these architectural advances, multi-agent and multi-step executions introduce complex dependencies through which errors may propagate across dependent steps or remain unresolved in repeated interactions, making the decisive error difficult to identify.

Failure Attribution in LLM-Based Multi-Agent Systems

The Who&When benchmark formalizes failure attribution as identifying the decisive agent-step pair in failed execution trajectories (Zhang et al. 2025d). Existing MAS attribution approaches can be broadly categorized into direct inference, replay-based, fine-tuning-based, and structure-guided methods. Direct LLM-based methods infer attribution targets directly from textual trajectories (Zhang et al. 2025d; Banerjee, Nair, and Borogovac 2025; Sun et al. 2026), but often struggle to capture fine-grained dependencies. Replay-based methods such as FAMAS (Ge et al. 2025) estimate responsibility through repeated executions, incurring substantial computational costs. Fine-tuning-based approaches such as AgenTracer and GraphTracer (Zhang et al. 2025a,b) learn specialized attribution models but require additional supervision and training resources (Qin et al. 2025; Wang et al. 2024).

Structure-guided methods are most closely related to EMFA. CHIEF (Wang et al. 2026) exploits hierarchical causal graphs to rank candidate errors, while FALAT (Rafi et al. 2026) incorporates propagation analysis after candidate pruning. However, these approaches do not explicitly model complete error propagation before candidate selection, which may lead to incomplete propagation paths or confusion between decisive errors and downstream symptoms. In contrast, EMFA constructs candidate sets from explicitly modeled error-propagation structures by tracing failure-related dependencies and analyzing both cascading propagation and persistent interaction loops. It then applies counterfactual verification to identify the decisive error.

Problem Formulation

To remain compatible with the Who&When benchmark and follow its established formulation, we adopt a problem setting consistent with prior work (Zhang et al. 2025d; Wang et al. 2026).

LLM-Based MAS. We consider a turn-based LLM multi-agent system in which exactly one agent acts at each execution step. Let $\mathcal{N}$ denote the set of agents. Given a task query Q , the system produces an execution trajectory

$$\tau = (x_1, x_2, \dots, x_L), \quad x_i = (a_i, u_i), \quad (1)$$

where $i \in \{1, \dots, L\}$ is the step index, $a_i \in \mathcal{N}$ is the agent acting at step i , and u_i denotes the action, observation, reasoning, or result produced at that step. Let $Z(\tau) \in \{0, 1\}$ denote the task outcome, where $Z(\tau) = 1$ indicates failure and $Z(\tau) = 0$ indicates success.

Final Failure Symptom. For a failed trajectory τ , let i_{final} denote the index of the final failure step $x_{\text{final}} = x_{i_{\text{final}}}$, at which the unsuccessful outcome becomes observable, e.g., through an incorrect final answer, premature termination, or an unresolved task requirement. We refer to the observable task-level mismatch at x_{final} as the *final failure symptom*. This step need not coincide with the decisive step, because the observed symptom may be a downstream consequence of an earlier error.

Failure Attribution. Given a failed trajectory τ with $Z(\tau) = 1$, failure attribution aims to identify the *decisive agent-step pair*. For each step i and its acting agent a_i , let $\tilde{\tau}^{(a_i, i)}$ denote the counterfactual execution obtained by correcting the action at step i and allowing the subsequent execution to unfold accordingly. We define the decisive-error indicator as

$$\Delta_{a_i, i}(\tau) = \begin{cases} 1, & \text{if } Z(\tilde{\tau}^{(a_i, i)}) = 0, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Thus, (a_i, i) is a *decisive error* if correcting the action at step i would recover the failed execution.

A failed trajectory may contain multiple agent-step pairs satisfying this counterfactual criterion. Following the benchmark annotation protocol, we take the earliest decisive error as the attribution target:

$$i^* = \min \{i \mid \Delta_{a_i, i}(\tau) = 1\}, \quad a^* = a_{i^*}. \quad (3)$$

Here, i^* is the index of the decisive step, a^* is the decisive agent, and x_{i^*} is the corresponding complete execution step. The objective is therefore to construct an attribution function

$$F(Q, \tau) = (a^*, i^*), \quad (4)$$

that recovers the ground-truth decisive agent-step pair from the task query and the observed failed trajectory.

Method

Our method, **Error-Propagation Modeling for Failure Attribution (EMFA)**, identifies the decisive error by tracing how the final failure symptom arises from earlier errors in a failed multi-agent execution. As illustrated in Figure 2, EMFA consists of five stages. First, *Final Failure Step Identification* locates the step at which the failure symptom is exposed. Second, *Structured Trajectory Construction* converts the flat execution log into a structured representation of agents, steps, and their dependencies. Third, *Error Propagation Modeling* captures both error propagation across dependent steps and unresolved errors that persist through repetitive interaction loops. Fourth, *Dual-Branch Candidate Screening* extracts candidate errors from the propagation and loop structures. Finally, *Counterfactual Decisive Error Verification* evaluates these candidates and identifies the decisive agent-step pair.

Final Failure Step Identification

Errors may arise early in a multi-agent execution but become observable only when the system produces an incorrect answer or terminates without resolving the task. EMFA therefore first identifies the *final failure step*, denoted as x_{final} , at

which the final failure symptom is exposed. This step serves as the terminal anchor for subsequent dependency tracing.

EMFA first extracts a task-success specification C_{success} from the query Q . The specification describes the expected outcome and any explicit requirements, such as source, format, temporal, or numerical constraints. It is derived solely from the query so that the evaluation criteria remain independent of the failed execution.

For a trajectory that produces an explicit final answer, EMFA scans the last w steps in reverse order and selects the latest agent step that directly answers the query, excluding intermediate plans, tool outputs, and progress updates. We set $w = 5$ to cover the terminal response region. If the trajectory reaches the maximum execution length without producing an explicit answer, its last step is selected instead, as it marks where the unresolved execution terminates. The selected step is evaluated against C_{success} to characterize the final failure symptom.

The final failure step is used only as an observation anchor. It may be a downstream manifestation of an earlier error and therefore does not necessarily coincide with either the decisive error or a propagation source.

Structured Trajectory Construction

A raw execution trajectory records the temporal order of agent actions but does not explicitly reveal how information is produced, transferred, and reused across agents and subtasks. EMFA therefore converts the flat trajectory into a two-level dependency representation

$$\mathcal{H} = (\mathcal{X}, \mathcal{S}, \mathcal{E}_{\text{step}}, \mathcal{E}_{\text{sub}}), \quad (5)$$

where $\mathcal{X}$ contains the trajectory steps together with their agent identities and subtask assignments, and $\mathcal{S} = \{S_1, \dots, S_M\}$ is a sequence of contiguous, non-overlapping semantic subtasks.

Within each subtask, $\mathcal{E}_{\text{step}}$ captures directed producer-consumer information dependencies between execution steps, with particular emphasis on information transferred across agents. An edge $(x_i, x_j) \in \mathcal{E}_{\text{step}}$ indicates that information produced at x_i is subsequently consumed at x_j , and records the information transferred between them. While most such edges represent inter-agent communication, dependencies between steps of the same agent are retained when they are necessary to reconstruct how task-relevant information is carried forward.

Across subtasks, $\mathcal{E}_{\text{sub}}$ connects an earlier subtask to a later one that reuses its outputs, including dependencies between non-adjacent subtasks. The subtask-level structure provides a coarse tracing scope, while the step-level dependencies support fine-grained producer identification during subsequent backward tracing.

We additionally apply deterministic post-processing to recover explicit local interactions that may be omitted during semantic extraction. When an instruction is immediately followed by a response from the addressed agent, EMFA adds the corresponding instruction-response dependency if it is absent. This rule completes clear interaction links without treating temporal adjacency alone as evidence of information dependency.

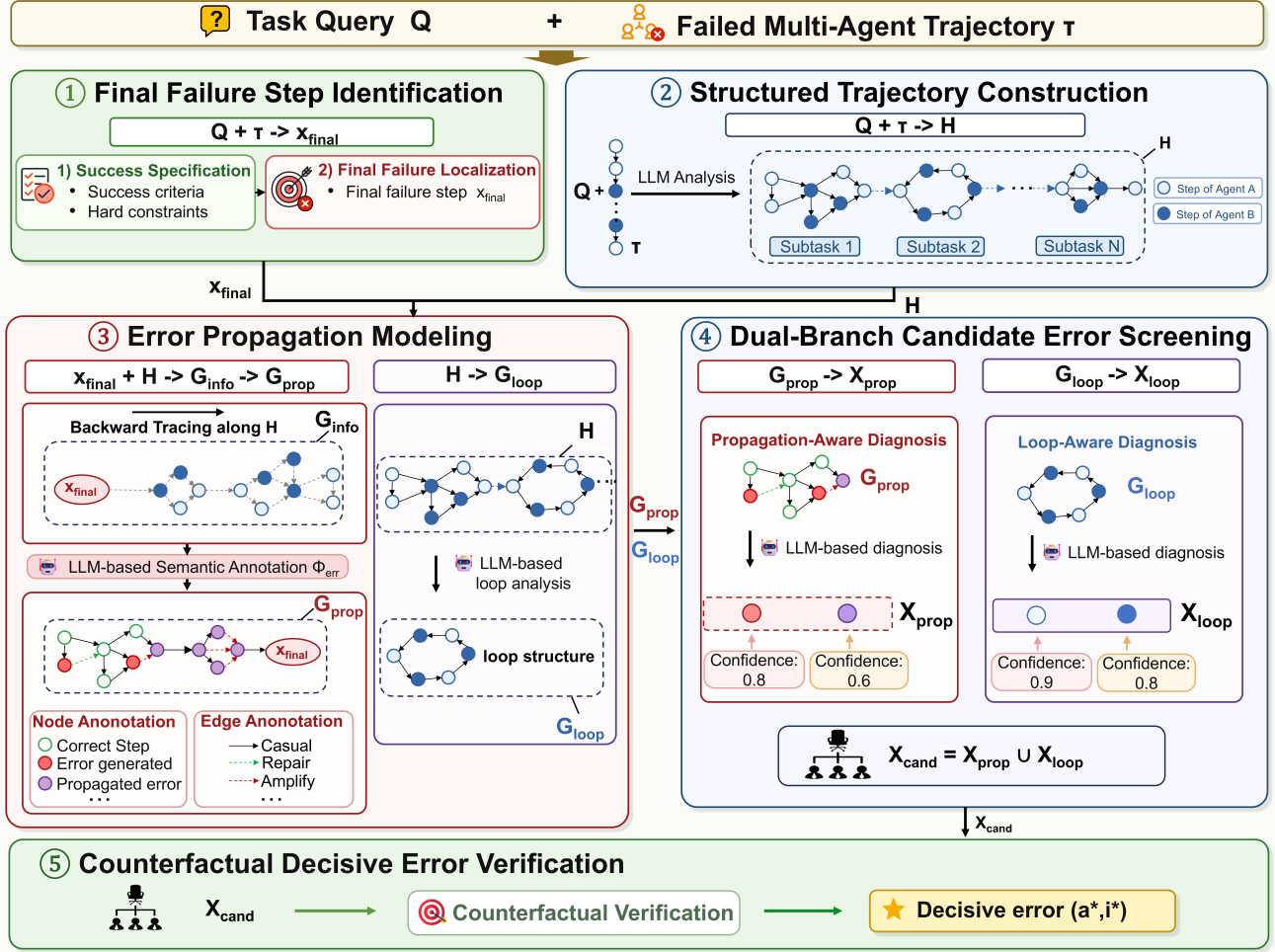


Figure 2: Overview of EMFA. Given a query Q and a failed trajectory τ , EMFA locates the final failure step corresponding to the final failure symptom and structures the trajectory. It then traces failure-related information backward, reconstructs how errors propagate chronologically using an LLM, screens candidate errors through propagation and loop branches, and applies counterfactual decisive error verification to select the decisive error step x^* and its responsible agent a^* .

Error Propagation Modeling

The structured trajectory $\mathcal{H}$ captures broad execution dependencies, many of which may be unrelated to the final failure symptom. EMFA therefore extracts two complementary failure-related structures: an error-propagation graph $\mathcal{G}_{\text{prop}}$, which describes how a relevant deviation propagates along information dependencies, and a persistent-loop graph $\mathcal{G}_{\text{loop}}$, which captures unresolved repetitive interactions.

Backward Relevance Tracing. Starting from the final failure step x_{final} , EMFA traverses the dependency structure in $\mathcal{H}$ backward to collect upstream subtasks and execution steps whose outputs are directly or indirectly reused by the current step. The reached steps and their producer-consumer dependencies form the relevant information graph

$$\mathcal{G}_{\text{info}} = (\mathcal{V}_{\text{info}}, \mathcal{E}_{\text{info}}). \quad (6)$$

Although $\mathcal{G}_{\text{info}}$ is constructed through backward traversal, its edges preserve the actual producer-to-consumer direction.

Thus, an edge $x_i \rightarrow x_j$ indicates that information produced at x_i is subsequently used at x_j . This process isolates the information channels potentially related to the observed failure. The detailed backward relevance tracing algorithm is provided in the technical supplement.

Forward Propagation Annotation. EMFA then processes the steps in $\mathcal{G}_{\text{info}}$ in chronological order and uses an LLM to annotate how task-relevant information changes along its nodes and edges. The annotations record whether a deviation is introduced, propagated, transformed, or corrected, thereby distinguishing propagation sources, propagation carriers, recovered deviations, unrecovered deviations, and the failure symptom exposed at x_{final} . The resulting graph is

$$\mathcal{G}_{\text{prop}} = (\mathcal{G}_{\text{info}}, \Phi_{\text{err}}), \quad (7)$$

where Φ_{err} denotes the semantic propagation annotations. Simple deterministic checks for explicit or omitted required

actions complement the LLM annotations. A propagation source describes the origin of a reconstructed propagation path and does not necessarily coincide with the decisive error.

Persistent Loop Extraction. Error propagation along producer–consumer dependencies does not capture failures in which agents repeatedly pursue the same unresolved objective without making substantive progress. EMFA therefore uses an LLM to identify persistent interaction loops directly from $\mathcal{H}$ by comparing the objectives, actions, observations, and progress of repeated interaction rounds. The resulting $\mathcal{G}_{\text{loop}}$ connects steps that repeatedly address the same unresolved objective without resolving it. Because such loops may span multiple subtasks and need not follow an ordinary information-dependency path, they are extracted independently of $\mathcal{G}_{\text{info}}$.

Together, $\mathcal{G}_{\text{prop}}$ and $\mathcal{G}_{\text{loop}}$ provide complementary evidence for subsequent candidate screening.

Dual-Branch Candidate Error Screening

EMFA independently screens candidate errors from the error-propagation and persistent-loop structures, retaining complementary causal explanations for subsequent counterfactual decisive error verification.

Propagation-Branch Screening. Given $\mathcal{G}_{\text{prop}}$, an LLM selector compares the reconstructed propagation events and returns a propagation candidate set $\mathcal{X}_{\text{prop}}$ containing at most two agent–step pairs. It prioritizes the earliest unrecovered deviation that materially affects subsequent reasoning or actions. Unless a later incorrect step introduces a new error, substantially transforms an existing one, or makes a consequential control decision, it is treated as a propagation carrier, or as the failure symptom when it coincides with x_{final} . Rule-based checks further clarify recurring responsibility boundaries, including plan author versus executor, faulty transformation versus raw evidence, and premature completion versus an earlier recoverable limitation.

Loop-Branch Screening. Each loop in $\mathcal{G}_{\text{loop}}$ is analyzed independently to identify the earliest step that initiates or sustains the unresolved repetition. The analysis distinguishes, when applicable, between an ambiguous or incorrect instruction and a failure to follow or correctly execute a valid instruction, while also considering omitted actions and failures to adapt after repeated unsuccessful attempts. When the same unresolved error recurs across multiple rounds, its first occurrence is selected as the group-level candidate. An aggregate selector retains at most two loop candidates in $\mathcal{X}_{\text{loop}}$.

The final candidate pool is the deduplicated union

$$\mathcal{X}_{\text{cand}} = \mathcal{X}_{\text{prop}} \cup \mathcal{X}_{\text{loop}}. \quad (8)$$

Each branch contributes at most two candidates, so full EMFA evaluates up to four candidates, while each single-branch variant evaluates up to two.

Counterfactual Decisive Error Verification

EMFA performs model-based counterfactual assessment rather than environment replay, consistent with the expert-

judgment annotation protocol of Who&When. For each candidate $(a_i, i) \in \mathcal{X}_{\text{cand}}$, the verifier assesses whether minimally repairing the step would prevent the downstream failure and satisfy C_{success} . A candidate is considered more causally decisive when its repair blocks the downstream failure process rather than merely correcting a later propagation carrier or the final symptom.

When multiple candidate repairs appear capable of recovering the execution, EMFA resolves the remaining ambiguity using predefined responsibility-boundary principles that distinguish the step responsible for introducing an error from downstream steps that merely execute, propagate, or expose it. Integrating counterfactual recovery with these principles, the verifier returns

$$(\hat{a}, \hat{i}) = F_{\text{cf}}(Q, \tau, C_{\text{success}}, \mathcal{X}_{\text{cand}}), \quad (9)$$

where $(\hat{a}, \hat{i})$ is EMFA’s predicted decisive agent–step pair. Here, $\hat{i}$ denotes the predicted index of the earliest decisive step, and $\hat{a}$ is the agent acting at that step. The prediction is evaluated against the ground-truth pair (a^*, i^*) .

Experiment Setup

Dataset

We evaluate EMFA on the *Who&When* benchmark (Zhang et al. 2025d), which annotates the decisive error step (“When”) and responsible agent (“Who”) for failed multi-agent trajectories. The test set contains two subsets: (1) **Hand-Crafted**, consisting of 58 logs from Magentic-One (Fourney et al. 2024), and (2) **Algorithm-Generated**, consisting of 126 logs generated from diverse architectures by CaptainAgent (Song et al. 2024). Tasks are derived from GAIA (Mialon et al. 2023) and AssistantBench (Yoran et al. 2024). Each sample provides an execution trajectory with agent identifiers and step-level outputs, together with annotated failure steps, responsible agents, and failure reasons. Ground-truth annotations are obtained through multi-round expert consensus.

Baselines

Given the emerging nature of MAS failure attribution, we compare EMFA against ten representative approaches spanning four paradigms. **Random** randomly selects an agent–step pair as a lower-bound baseline. **LLM-based Prompting** includes All-at-Once, Step-by-Step, and Binary Search from Who&When (Zhang et al. 2025d), together with ECHO (Banerjee, Nair, and Borogovac 2025) and SDBL (Sun et al. 2026), which directly infer or narrow attribution targets from execution trajectories. **Fine-tuning-based Methods** include AgenTracer and GraphTracer (Zhang et al. 2025a,b), which learn attribution models from additional supervision. **Structure-guided Methods** include CHIEF (Wang et al. 2026) and FALAT (Rafi et al. 2026), which exploit trajectory structures or propagation information for attribution. We exclude FAMAS (Ge et al. 2025) because it requires access to task ground truth $\mathcal{G}$, whereas all compared methods operate without $\mathcal{G}$.

Table 1: Main results on the Who&When benchmark. Each cell reports accuracy (%). Best results in each column are in **bold**; second-best results are underlined. For EMFA, values in parentheses indicate the gap to the second-best result when EMFA ranks first, or to the best result otherwise.

Type	Method	Hand-Crafted Dataset		Algorithm-Generated Dataset	
		Agent_ACC	Step_ACC	Agent_ACC	Step_ACC
Heuristic	Random	12.00	4.20	29.10	19.10
LLM-based Prompting	All-at-Once	48.28	5.17	59.52	15.87
	Step-by-Step	34.30	6.90	28.30	17.80
	Binary Search	36.20	6.90	30.10	16.60
	ECHO	67.90	26.80	67.20	27.20
	SDBL	68.52	27.78	63.49	39.68
Fine-tuning-based	AgenTracer	63.82	20.68	63.73	37.30
	GraphTracer	69.74	27.97	<u>67.42</u>	44.35
Structure-guided	CHIEF	72.41	<u>29.31</u>	68.80	<u>45.60</u>
	FALAT	<u>73.00</u>	29.10	59.70	38.00
EMFA (Ours)		74.14 (+1.14)	32.76 (+3.45)	66.67 (-2.13)	50.00 (+4.40)

Evaluation Metrics

Following prior work (Zhang et al. 2025d,a), we employ two evaluation metrics: **agent-level accuracy** and **step-level accuracy**. Agent-level accuracy measures whether a method correctly identifies the responsible failure agent, while step-level accuracy measures whether it correctly identifies the exact decisive error step. All results use a strict Top-1 criterion. EMFA results are averaged over three independent runs, while published baseline values follow the reporting protocols of their original papers. While the ground-truth answer to each query is provided in the benchmark, real-world failure attribution often lacks access to such answers. Thus, unless otherwise specified, all experiments in this paper are conducted without using task ground truth $\mathcal{G}$.

Implementation Details

We implement EMFA in Python 3.11 on Ubuntu. Unless otherwise specified, the base LLM is GPT-4o, with API temperature set to 0, a maximum generation length of 4,096 tokens. For baselines without complete public implementations, we report published results when they exceed our applicable GPT-4o reproductions, thereby avoiding underestimation of baseline performance. Additionally, detailed prompts, configurations, and further cost and qualitative analyses are provided in the supplementary materials.

Results

Main Results

As shown in Table 1, EMFA achieves the best step-level attribution accuracy on both subsets of the Who&When benchmark. It improves the previous best results, obtained by CHIEF, by 3.45 and 4.40 percentage points on the Hand-Crafted and Algorithm-Generated subsets, respectively, while substantially outperforming direct prompting baselines. Across all methods, step-level accuracy is markedly lower than agent-level accuracy, highlighting the



Figure 3: Agent- and step-level accuracy of All-at-Once, Step-by-Step, CHIEF (SOTA), and EMFA across six log-length tiers on the Hand-Crafted subset.

greater difficulty of precisely locating the decisive step. EMFA’s consistent gains on this stricter metric across both subsets demonstrate its advantage in fine-grained failure attribution. At the agent level, EMFA achieves the best result on the Hand-Crafted subset and remains competitive on the Algorithm-Generated subset.

Effect of Trajectory Length on Failure Attribution

Following the protocol of Sun et al. (2026), we evaluate agent and step accuracy across different log lengths on the **Hand-Crafted** subset, as longer trajectories typically increase attribution ambiguity by enlarging the distance between decisive

Table 2: EMFA performance on the Who&When benchmark using various backbone LLMs. Agent and Step denote Agent_ACC and Step_ACC, respectively. Best results in each column are in **bold**.

LLM Backbones	Hand-Crafted		Alg.-Generated	
	Agent	Step	Agent	Step
DeepSeek-V4-Flash	61.40	24.56	55.56	40.48
Qwen3.5-397B-A17B	72.41	29.31	56.35	39.68
GLM-4.7	67.24	25.86	53.59	40.48
GPT-4o	74.14	32.76	66.67	50.00

Table 3: Ablation study of dual-branch candidate construction in EMFA. Each active branch contributes at most two candidates, while counterfactual verification remains unchanged.

EMFA Variants	Hand-Crafted		Alg.-Generated	
	Agent	Step	Agent	Step
Direct LLM Screening	53.45	18.97	42.06	29.35
w/o Propagation Branch	27.59	1.72	21.43	13.49
w/o Loop Branch	55.17	31.03	44.44	33.33
EMFA	74.14	32.76	66.67	50.00

errors and final failure symptoms. Test samples are grouped into six tiers by step count: Level 1 (5–16 steps, $n=9$), Level 2 (17–21, $n=10$), Level 3 (24–32, $n=10$), Level 4 (33–53, $n=9$), Level 5 (59–106, $n=11$), and Level 6 (113–130, $n=9$). We compare All-at-Once, Step-by-Step, the previous SOTA structure-guided method CHIEF, and EMFA under this stratification; CHIEF results are from our reproduction with Claude-Opus-4.6-thinking.

Performance across different trajectory lengths reveals a trade-off between fine-grained localization and agent-level attribution. EMFA maintains relatively strong step-level accuracy across the six tiers, suggesting that propagation-aware reasoning remains effective on longer execution traces. However, EMFA and CHIEF show lower agent-level accuracy on some longer trajectories, one possible explanation is that structure-guided methods introduce stronger inductive biases and may miss failure patterns not represented by their structures: EMFA focuses on propagation and persistent interaction patterns, while CHIEF relies on predefined agent dependencies. Such biases may occasionally miss responsible agents when failures arise from unmodeled failure propagation patterns. Nevertheless, step-level attribution remains the more fine-grained objective, and EMFA’s consistent advantage highlights the effectiveness of explicit error propagation modeling. Future work will explore additional failure propagation patterns to further improve attribution robustness.

Further Analysis

We conduct further analyses to understand the robustness and effectiveness of EMFA beyond the main benchmark results, including backbone robustness and ablation studies.

Additional qualitative analysis and inference-cost analysis are provided in the technical supplement.

Performance on Different LLMs. As shown in Table 2, EMFA remains effective across all tested backbones, although its absolute performance varies with the underlying LLM. GPT-4o achieves the best results on both subsets, with a particularly clear advantage on Algorithm-Generated step-level attribution (50.00% versus 39.68–40.48% for the open-weight models). Among the open-weight backbones, Qwen3.5-397B-A17B performs best on the Hand-Crafted subset, while the three models achieve comparable step-level accuracy on the Algorithm-Generated subset. Overall, these results indicate that EMFA can be instantiated with different base LLMs, while backbone selection continues to affect attribution performance.

Ablation Study. We ablate the dual-branch candidate-screening stage while keeping Counterfactual Decisive Error Verification fixed. Each retained branch contributes at most two candidates. As shown in Table 3, full EMFA consistently outperforms direct LLM screening and both single-branch variants across all evaluation metrics. Removing the Propagation Branch causes a substantial performance degradation, particularly in step-level accuracy, showing that propagation-aware candidates contribute strongly to the full system. In contrast, removing the Loop Branch results in a smaller drop on the Hand-Crafted subset but causes a pronounced degradation on the Algorithm-Generated subset, where step-level accuracy decreases from 50.00% to 33.33%. This indicates that loop-aware screening provides complementary benefits by identifying failures caused by persistent unproductive interactions.

Overall, the full dual-branch design provides the strongest end-to-end attribution performance, indicating complementary candidate coverage from propagation and loop analysis.

Conclusion and Limitations

This paper presents EMFA, a failure attribution framework for LLM-based multi-agent systems that models cascading error propagation and persistent interaction loops to identify decisive errors. Rather than localizing observable failure symptoms, EMFA traces failure-related dependencies, models error propagation through propagation and persistent interaction patterns, and applies counterfactual verification to identify decisive agent-step pairs. Experiments on the Who&When benchmark demonstrate that EMFA achieves state-of-the-art step-level attribution accuracy while remaining competitive at the agent level.

Despite its effectiveness, EMFA introduces additional inference-time overhead due to explicit trajectory structuring and propagation modeling. Future work will explore more efficient propagation modeling strategies to reduce computational costs while preserving attribution reliability.

References

Banerjee, A.; Nair, A.; and Borogovac, T. 2025. Where Did It All Go Wrong? A Hierarchical Look into Multi-Agent Error Attribution. arXiv:2510.04886.

- Cemri, M.; Pan, M. Z.; Yang, S.; Agrawal, L. A.; Chopra, B.; Tiwari, R.; Keutzer, K.; Parameswaran, A.; Klein, D.; Ramchandran, K.; Zaharia, M.; Gonzalez, J. E.; and Stolica, I. 2025. Why Do Multi-Agent LLM Systems Fail? arXiv:2503.13657.
- Fourney, A.; Bansal, G.; Mozannar, H.; Tan, C.; Salinas, E.; Niedtner, F.; Proebsting, G.; Bassman, G.; Gerrits, J.; Alber, J.; et al. 2024. Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks. arXiv:2411.04468.
- Ge, Y.; Xie, L.; Li, Z.; Pei, Y.; and Zhang, T. 2025. Who Is Introducing the Failure? Automatically Attributing Failures of Multi-Agent Systems via Spectrum Analysis.
- Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Zhang, C.; Wang, J.; Tang, Z.; Yau, S. K. S.; Lin, Z.; Zhou, L.; Ran, C.; Xiao, L.; Wu, C.; and Schmidhuber, J. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework.
- Mialon, G.; Fourier, C.; Swift, C.; Wolf, T.; LeCun, Y.; and Scialom, T. 2023. GAIA: A Benchmark for General AI Assistants. arXiv:2311.12983.
- Phan, H. N.; Nguyen, T. N.; Nguyen, P. X.; and Bui, N. D. Q. 2025. HyperAgent: Generalist Software Engineering Agents to Solve Coding Tasks at Scale. arXiv:2409.16299.
- Qian, C.; Liu, W.; Liu, H.; Chen, N.; Dang, Y.; Li, J.; Yang, C.; Chen, W.; Su, Y.; Cong, X.; Xu, J.; Li, D.; Liu, Z.; and Sun, M. 2024. ChatDev: Communicative Agents for Software Development. arXiv:2307.07924.
- Qin, Y.; Wang, S.; Lou, Y.; Dong, J.; Wang, K.; Li, X.; and Mao, X. 2025. Soap FL: A Standard Operating Procedure for LLM-based Method-Level Fault Localization. *IEEE Transactions on Software Engineering*.
- Rafi, M. N.; Ahasanuzzaman, M.; Kim, D. J.; Wang, Z.; and Chen, T.-H. 2026. FALAT: Tracing Failures in LLM Agent Trajectories via Dependency-Guided Search.
- Song, L.; Liu, J.; Zhang, J.; Zhang, S.; Luo, A.; Wang, S.; Wu, Q.; and Wang, C. 2024. Adaptive In-conversation Team Building for Language Model Agents. arXiv:2405.19425.
- Sun, K.; Li, W.; Dong, B.; Lin, Y.; Zhang, J.; and Shi, B. 2026. Scope Delineation Before Localization: A Two-Stage Framework for Enhancing Failure Attribution in Multi-Agent Systems. *AAAI2026*.
- Wang, X.; Xu, Q.; Xu, K.; Yu, T.; Ding, B.; Feng, D.; and Dou, Y. 2024. Large pretrained foundation model for key performance indicator multivariate time series anomaly detection. *IEEE Open Journal of the Computer Society*.
- Wang, Y.; Wu, W.; Wang, J.; and Wang, Q. 2026. From Flat Logs to Causal Graphs: Hierarchical Failure Attribution for LLM-based Multi-Agent Systems. arXiv:2602.23701.
- Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; Awadallah, A. H.; White, R. W.; Burger, D.; and Wang, C. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.
- Yoran, O.; Amouyal, S. J.; Malaviya, C.; Bogin, B.; Press, O.; and Berant, J. 2024. AssistantBench: Can Web Agents Solve Realistic and Time-Consuming Tasks? arXiv:2407.15711.
- Zhang, G.; Wang, J.; Chen, J.; Zhou, W.; Wang, K.; and Yan, S. 2025a. AgenTracer: Who Is Inducing Failure in the LLM Agentic Systems?
- Zhang, H.; Shi, Y.; Gu, X.; You, H.; Zhang, Z.; Gan, L.; Yuan, Y.; and Huang, J. 2025b. GraphTracer: Graph-Guided Failure Tracing in LLM Agents for Robust Multi-Turn Deep Search.
- Zhang, J.; Xiang, J.; Yu, Z.; Teng, F.; Chen, X.; Chen, J.; Zhuge, M.; Cheng, X.; Hong, S.; Wang, J.; Zheng, B.; Liu, B.; Luo, Y.; and Wu, C. 2025c. AFlow: Automating Agentic Workflow Generation. arXiv:2410.10762.
- Zhang, S.; Yin, M.; Zhang, J.; Liu, J.; Han, Z.; Zhang, J.; Li, B.; Wang, C.; Wang, H.; Chen, Y.; and Wu, Q. 2025d. Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems. arXiv:2505.00212.