Towards Self-Improving Error Diagnosis in Multi-Agent Systems

Jiazheng Li[1*] **Emine Yilmaz**[2,3*] **Bei Chen**[2] **Dieu-Thu Le**[2]
[1]King's College London [2]Amazon Alexa AI [3]University College London
jiazheng.li@kcl.ac.uk {eminey,chenbe,deule}@amazon.com

Abstract

Large Language Model (LLM)-based Multi-Agent Systems (MAS) enable complex problem-solving but introduce significant debugging challenges, characterized by long interaction traces, inter-agent dependencies, and delayed error manifestation. Existing diagnostic approaches often rely on expensive expert annotation or "LLM-as-a-judge" paradigms, which struggle to pinpoint decisive error steps within extended contexts. In this paper, we introduce ERRORPROBE, a self-improving framework for semantic failure attribution that identifies responsible agents and the originating error step. The framework operates via a three-stage pipeline: (1) operationalizing the MAS failure taxonomy to detect local anomalies, (2) performing symptom-driven backward tracing to prune irrelevant context, and (3) employing a specialized multi-agent team (Strategist, Investigator, Arbiter) to validate error hypotheses through tool-grounded execution. Crucially, ERRORPROBE maintains a verified episodic memory that updates only when error patterns are confirmed by executable evidence, without the need for annotation. Experiments across the TRACERTRAJ and WHO&WHEN benchmarks demonstrate that ERRORPROBE significantly outperforms baselines, particularly in step-level localization, while the verified memory enables robust cross-domain transfer without retraining.

1 Introduction

Large language model (LLM)-based multi-agent systems (MAS) have demonstrated strong performance across diverse domains, from software engineering (Hong et al., 2024; Qian et al., 2024) and web navigation (Yao et al., 2022a,b) to scientific reasoning (Gottweis et al., 2025), education (Li et al., 2025b) and tool use (Schick et al., 2023; Patil et al., 2024). Their core appeal is simple: by decomposing a complex task into multiple steps and assigning specialized roles (e.g., architect, engineer, tester) (Li et al., 2024; Wu et al., 2024; OpenAI, 2024; Yan et al., 2025), MAS can solve problems that often exceed the capacity of a single LLM call.

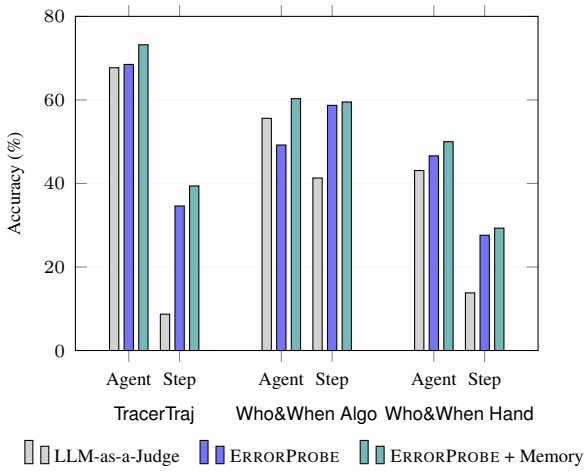

Figure 1: Claude 3.7 Sonnet results on agent attribution and decisive-step localization. ERRORPROBE substantially improves step-level localization over LLM-as-a-Judge while maintaining competitive agent accuracy. Adding verified memory retains strong performance while enabling reuse across tasks without LLM training.

This increased power of MAS creates a practical debugging problem (Zhou et al., 2024; Cemri et al., 2025; Zhang et al., 2026, 2025b). Consider a common failure pattern: an architect introduces a subtle specification error, an engineer faithfully implements it, and a tester verifies the wrong behavior. The run may only fail many turns later, after multiple agents have built on the initial mistake. When this happens, practitioners face a central diagnostic question: "*which agent caused the failure, and at what step did the error originate?*"

In this paper, we study *failure attribution* in MAS–given a multi-agent interaction trace and a failure symptom (e.g., a final wrong answer, an exception, or unmet constraints), the goal is to identify the responsible agent(s) and the originating

[*]Work done at Amazon Alexa AI.

Findings of the Association for Computational Linguistics: ACL 2026, pages 2063–2077

turn/step (and optionally the failure type). This problem is challenging for several reasons unique to multi-agent settings:

- **Long interaction traces:** Production MAS runs can span dozens to over a hundred turns, often exceeding practical context windows (Zhang et al., 2026; Packer et al., 2023; Liu et al., 2024).
- **Delayed manifestation:** Early mistakes may only surface much later, complicating causal attribution (Zhang et al., 2026, 2025b).
- **Inter-agent dependencies:** Each agent conditions on prior agents' outputs, creating long (and sometimes branching) causal chains (Wu et al., 2024).
- **Diverse failure modes:** Failures range from specification errors to coordination breakdowns and verification gaps, each requiring different diagnostics (Cemri et al., 2025).

Prior work on failure analysis in LLM-based MAS largely falls into three directions: First, taxonomy-driven studies such as the Multi-Agent System Taxonomy (MAST) (Cemri et al., 2025) provide a principled lens over diverse failure modes, but constructing and labeling failures requires substantial expert human annotation effort, which is expensive and difficult to scale to the volume and length of real MAS traces (Cemri et al., 2025). Second, specialized tracers trained on curated supervision can improve attribution performance, but they depend on costly data generation pipelines and may require continual retraining or adaptation as agent frameworks, trace distributions, and error types evolve (Zhang et al., 2026). Third, prompting a general LLM as a judge is attractive for its low overhead and broad applicability, yet current benchmarks show that judge-style methods remain far from reliable for task-specific attribution. Especially for pinpointing the decisive error step in long, delayed-failure traces (Zhang et al., 2025b; Ma et al., 2025). Even training-free transfer mechanisms for judge localization can be sensitive to domain shift (Yu et al., 2026). These limitations motivate a self-evolving multi-agent diagnosis framework. By emulating human expert debugging: decomposing the diagnosis into specialized roles for backward tracing, hypothesis generation, and tool-grounded verification, such a framework can manage the complexity of long interaction traces and improve attribution accuracy without sacrificing generality across evolving failure modes (Zhuge et al., 2025).

To tackle the above issues, we present **ERRORPROBE**, a MAS framework for failure attribution. Given the raw interaction history (including inter-agent messages and tool outputs) and a description of the failure symptom (e.g., an exception or incorrect final answer), our framework identifies the responsible agent(s) and the earliest error-inducing step. ERRORPROBE performs symptom-driven backward tracing to selectively retrieve causally relevant earlier turns from long interactions, augments the analysis with structured semantic signals from a lightweight detector that operationalizes the MAST failure taxonomy, and maintains a verified memory of reusable error/patch patterns that is updated only when patterns are confirmed through executable evidence (e.g., successful reproduction via code sandboxes) and arbiter verification, improving robustness while avoiding noisy accumulation under distribution shift.

We evaluate ERRORPROBE on three benchmarks: TRACETRAJ (Zhang et al., 2026) and two splits of WHO&WHEN (Zhang et al., 2025b), spanning multiple multi-agent frameworks and task settings. Across baselines including LLM-as-a-judge and prior tracing-based approaches, ERRORPROBE improves both *agent-level* and *step-level* attribution, with ablations showing that backward tracing, structured failure-mode cues, and verified memory each contribute complementary gains. Finally, we demonstrate that our verified memory design successfully enables cross-domain transfer, allowing patterns learned in one domain to improve attribution in another without the degradation seen in naive approaches.

Our contributions are as follows:

- We introduce **ERRORPROBE**, a framework for semantic failure attribution in LLM-based multi-agent systems that localizes responsible agent(s) and originating error steps.
- We operationalize the **MAST** taxonomy (Cemri et al., 2025) into a lightweight detector that scans interaction traces for local anomalies (e.g., misalignment or verification gaps). These structural tags serve as heuristic priors to narrow the search space, providing interpretable, structured failure-mode signals to guide attribution.
- We demonstrate effective cross-domain transfer with verified memory, overcoming the brittleness of prior methods that often require retraining or struggle with distribution shift. We show that patterns learned in one dataset (e.g., KodCode) improve diagnosis in another (e.g., TracerTraj), highlighting the importance of verification-aware

memory management for robust generalization.

2 Related Work

Multi-Agent Error Attribution. A growing body of work addresses failure attribution in multi-agent systems, identifying responsible agents and error steps within interaction traces. Zhang et al. (2025b) formalize this problem, highlighting the difficulty of step-level localization. To enable scalable supervision, Zhang et al. (2026) introduce automated data construction via counterfactual replay, while subsequent methods explore spectrum-style analysis (Ge et al., 2025), hierarchical context decomposition (Banerjee et al., 2025), and communication efficiency (Zhang et al., 2025a). Complementary research focuses on characterizing failures through taxonomies (Cemri et al., 2025), root-cause diagnosis benchmarks (Ma et al., 2025), and human-annotated debugging datasets (Deshpande et al., 2025). Unlike prior methods that rely on heavy supervision, costly replay, or domain-sensitive caching (Yu et al., 2026), our work targets semantic diagnosis effective on long, delayed-failure traces by combining backward tracing with structured failure-mode signals.

LLM and Agent Evaluation. The "LLM-as-a-Judge" paradigm has become a standard for scalable evaluation (Zheng et al., 2023), often enhanced via chain-of-thought or explicit rubrics (Liu et al., 2023; Kim et al., 2024). While recent studies caution against potential biases in automated judges (Bavaresco et al., 2025; Ye et al., 2025), the paradigm has expanded to "Agent-as-a-Judge," which provides process-level feedback for agentic workflows (Zhuge et al., 2025). However, whereas standard judges typically produce scalar or pairwise rankings, our setting requires causal localization. We therefore move beyond free-form judging to trace-level backward reasoning augmented with semantic verification.

Self-Evolving Agents and Memory. Research on self-improving agents leverages verbal reinforcement and memory. Early approaches utilize iterative reflection (Madaan et al., 2023) or episodic buffers (Shinn et al., 2023) to refine outputs. To support lifelong learning, systems have adopted skill libraries (Wang et al., 2024) and memory management architectures (Park et al., 2023; Packer et al., 2023; Shen et al., 2025). Recent surveys provide comprehensive taxonomies of agent memory mechanisms across forms, functions, and dynamics (Zhang et al., 2024; Hu et al., 2025). Self-evolving frameworks integrate these components to adapt agent behavior at test-time (Liang et al., 2026; He et al., 2026; Liu et al., 2026). Addressing the challenge of memory corruption in these systems, we propose a verified-before-write update gate. This ensures that only patches supported by verification signals are committed, preventing drift while retaining reusable error patterns.

3 Problem Formulation

We formalize the problem of **Multi-Agent Failure Attribution** as a sequential prediction task. Let $\mathcal{T}$ be the space of interaction traces generated by a multi-agent system. These traces consist of a chronological sequence of turn-based interactions, where each step records the sender identity (agent), the recipient, and the semantic content (e.g., natural language messages, code blocks, or tool execution outputs). A task instance is defined as a tuple (x, y), where $x \in \mathcal{T}$ is a failed execution trace containing a sequence of messages and tool outputs $x = [m_1, m_2, \ldots, m_L]$, and $y = (a^*, t^*, f^*)$ is the ground-truth attribution tuple consisting of:

- The **culprit agent** a^* responsible for the root cause.
- The **decisive error step** $t^* \in [1, L]$, corresponding to the specific interaction m_{t^*} (e.g., a hallucinated instruction, incorrect parameter, or buggy code snippet) where the logic fault was first introduced, as opposed to where the failure symptom became visible.
- For the failure mode f^*, we specifically adopt the MAST taxonomy (Cemri et al., 2025). This framework organizes multi-agent failures into 14 error modes across three hierarchical families:
  - Specification Issues (FC1): Errors in the initial setup or prompts (e.g., Disobey Task Specification where constraints are violated, or Disobey Role Specification where an agent acts outside its assigned persona).
  - Misalignment (FC2): Failures in inter-agent coordination (e.g., Ignored Other Agent's Input where context is lost between turns, or Reasoning-Action Mismatch where the output contradicts the agent's internal thought process).
  - Verification Failures (FC3): Flaws in the quality assurance process (e.g., No/Incomplete Verification where a Tester

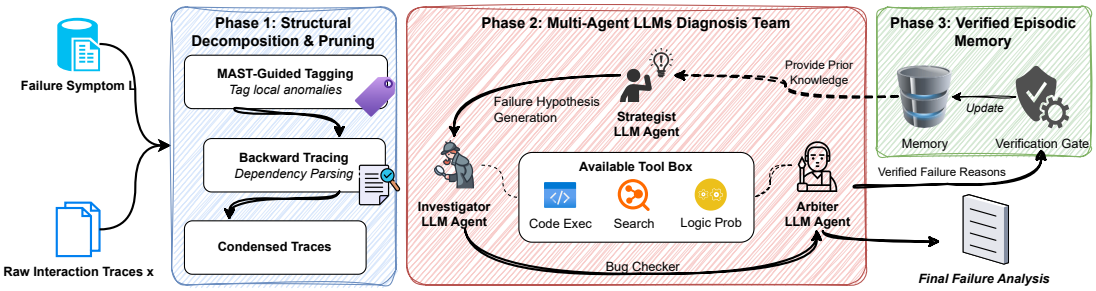

Figure 2: Overview of ERRORPROBE. The system prunes long traces via dependency parsing, then employs a Strategist-Investigator-Arbiter team to diagnose the root cause, updating memory only upon successful verification.

approves without testing, or Incorrect Verification where valid solutions are rejected).

Standard evaluation typically treats this as an i.i.d. classification problem, maximizing the likelihood $P(y|x)$. However, real-world debugging is a continuous process where the diagnosis system should learn from past failures. We therefore define a **Sequential Diagnosis** setting: the system maintains a mutable memory state $\mathcal{M}_i$ at step i. Given a stream of failure tasks $\{(x_i, y_i)\}_{i=1}^{N}$, the model predicts $\hat{y}_i = \mathcal{F}(x_i; \mathcal{M}_{i-1})$ and subsequently updates its state:

$$\mathcal{M}_i \leftarrow \text{Update}(\mathcal{M}_{i-1}, x_i, \hat{y}_i, \text{Verify}(\hat{y}_i)). \quad (1)$$

Our objective is to maximize cumulative attribution accuracy over the stream while minimizing memory size and false positive updates.

4 The ERRORPROBE Framework

To address the challenges of long contexts and delayed error manifestation, ERRORPROBE operates via a three-stage pipeline: (1) Structural Decomposition via the MAST taxonomy, (2) Symptom-Driven Backward Tracing to prune irrelevant context, and (3) a Self-Evolving Multi-Agent Diagnosis loop with verified memory. The overall architecture is illustrated in Figure 2.

4.1 MAST-Guided Structural Decomposition

Raw interaction traces are often unstructured and noisy. To provide semantic grounding, we operationalize the MAST taxonomy (defined in Section 3) into a lightweight signal detector. Instead of redefining the failure modes here, we focus on detecting their local symptoms.

Rather than asking the model to diagnose x directly, we first parse the trace to extract a structured representation $S_x = \{(\text{agent}_t, \text{role}_t, \text{action}_t)\}_{t=1}^{L}$. Role ($\text{role}_t$) is the assigned persona (e.g., *Architect*, *Software Engineer*, *Product Manager*) extracted from the system prompt. Action (action_t) is the semantic type of the turn (e.g., *Natural Language Instruction*, *Tool Call*, or *Execution Output*). We then apply a taxonomy-conditioned prompt to tag potential local anomalies. Unlike the global failure symptom provided as input (which indicates that the task failed), these tags identify where specific step-level deviations occurred (e.g., a 'Tool output ignored' warning at Step 5). These weak signals serve as heuristic priors for the subsequent reasoning agents, narrowing the search space from L steps to a smaller set of candidate regions.

4.2 Symptom-Driven Backward Tracing

A core challenge in MAS debugging is the distance between the root cause (e.g., a wrong parameter at $t = 5$) and the symptom (e.g., a crash at $t = 50$). Processing the full linear history often leads to "lost-in-the-middle" phenomena.

Therefore, we implement a **Backward Tracing** mechanism that reconstructs the causal chain of the failure. Given a failure symptom at step L:

1. **Dependency Parsing:** We construct a dependency graph $G = (V, E)$ where nodes are messages and edges represent information flow (e.g., Agent B cites Agent A's output).
2. **Recursive Pruning:** Starting from the symptom node v_L, we perform a breadth-first search on incoming edges to identify the *Effective Receptive Field* of the error.
3. **Context Compression:** We mask unconnected branches (e.g., parallel sub-tasks that succeeded), retaining only the causal lineage $x' \subset x$.

This condensed context x' focuses the model's attention on the agents and steps that actually influenced the failure, significantly reducing noise for the diagnosis agents.

4.3 Multi-Agent Diagnosis Architecture

The diagnosis is executed by a team of three specialized agents designed to mimic a human debugging workflow: hypothesis generation, verification, and adjudication.

The Strategist (Evaluation Lead). This agent acts as the high-level planner. It receives the condensed trace x' and the structural tags from Section 4.1. Its goal is to formulate a set of *Hypotheses* $H = \{(s_j, \text{suspected\_mode})\}$. To do this efficiently, the Strategist queries the long-term memory $\mathcal{M}$ (see Section 4.4) to retrieve k historical failure patterns. In 'cold start' scenarios (empty memory) or when no semantically similar patterns are found, the Strategist gracefully degrades to first-principles reasoning, relying solely on the structural MAST tags and trace context. As the system encounters and verifies new failures, this memory naturally populates, progressively shifting the Strategist from zero-shot reasoning to pattern-aware diagnosis.

The Investigator (Bug Checker). To combat the hallucination of error causes, the Investigator is strictly grounded in tool usage. For every hypothesis $h \in H$, it must produce *evidence* E_h. We provide a dedicated toolset:

- `CodeExec`: Re-runs code snippets in a sandbox to confirm runtime errors.
- `LogicProbe`: Verifies specific pre-/post-conditions (e.g., "Check if variable X in step t matches the constraints in step $t-5$").

The Investigator cannot simply state "The code is wrong"; it must generate a diff or an execution log proving the discrepancy.

The Arbiter (Verifier). The Arbiter serves as the final decision gate. It aggregates the hypotheses and their corresponding evidence sets $\{(h, E_h)\}$, filters out hypotheses where E_h is empty or inconclusive, and outputs the final prediction $\hat{y} = (a, t, f)$ with a confidence score $c \in [0, 1]$. The Arbiter also decides whether the diagnosed pattern is novel and robust enough to be committed to memory, acting as the safeguard against memory corruption.

4.4 Verified Episodic Memory

To enable self-improvement while mitigating the risk of memory corruption under distribution shift, ERRORPROBE maintains a dynamic memory $\mathcal{M}$ of verified failure patterns. Unlike naive caching, we enforce a **Strict Verification Gate** to ensure only robust diagnoses are retained.

Verification Gate. While the Arbiter evaluates evidence to render a verdict for the current task, the Verification Gate acts as a stricter filter for long-term retention. It ensures that only diagnoses with high-confidence, reproducible tool evidence are committed to memory. Formally, a new diagnosis $\hat{y}_t$ and its associated evidence E_t are committed to the memory state $\mathcal{M}_t$ if and only if $\text{Verify}(E_t) \wedge c_t > \tau$:

$$\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{(\hat{y}_t, E_t, \sigma_t)\} \tag{2}$$

where $\text{Verify}(E_t)$ represents the success of the Investigator's tool-grounded reproduction, c_t is the Arbiter's confidence, and τ is a set threshold. This prevents "hallucinated" error patterns from polluting the diagnostic history.

Verified Episodic Memory. To enable self-improvement while ensuring interpretability, ERRORPROBE maintains a dynamic memory $\mathcal{M}$ of verified failure patterns. Each entry $e \in \mathcal{M}$ is stored as a tuple $\langle \mathbf{s}_e, v_e, \sigma_e \rangle$. The structured signature $\mathbf{s}_e$ comprises interpretable features extracted from the trace, including the detected MAST anomaly (e.g., *step_repetition*), the specific `tool` or `api` endpoint involved, the abstracted argument types (`args`), and context slots (`ctx`) such as agent role or task domain. Associated with this signature is the diagnosis recipe v_e, containing the canonical error pattern and verification guard, and a set of meta-statistics σ_e that track recency, frequency, impact, and performance delta.

Relevance Scoring & Retrieval. When diagnosing a new trace x, we construct a query signature $\mathbf{s}_x$ and compute a relevance score that combines structural similarity with a quality-weighted RFI-Δ score:

$$S(x, e) = \underbrace{\text{Sim}(\mathbf{s}_x, \mathbf{s}_e)}_{\text{Structural Match}} \times \underbrace{S_{\text{RFI}}(e)}_{\text{Quality Score}} \tag{3}$$

where $\text{Sim}(\cdot)$ computes a weighted match over signature fields (requiring exact match on the MAST tag), and S_{RFI} is a composite of recency, frequency, impact, and performance delta. To address the cold start problem, we apply a strict similarity threshold $\tau_{\text{ret}} = 0.75$. If $\max_e \text{Sim}(x, e) < \tau_{\text{ret}}$, retrieval returns an empty set, and the Analyzer defaults to first-principles reasoning using only the MAST taxonomy.

Algorithm 1 Self-Evolving Diagnosis Routine

Require: Failed trace x_t, Memory $\mathcal{M}_{t-1}$
Ensure: Diagnosis $\hat{y}$, Updated Memory $\mathcal{M}_t$
Compression: $x'_t \leftarrow \text{BackwardTrace}(x_t, \text{MastPriors})$
Retrieval: $\mathcal{P} \leftarrow \text{Retrieve}(\mathcal{M}_{t-1}, x'_t, k)$
Plan: $H \leftarrow \text{Strategist}(x'_t, \mathcal{P})$
for hypothesis h in H **do**
    $E_h \leftarrow \text{Investigator}(h, \text{Tools})$
end for
Verdict: $(\hat{y}, c, E) \leftarrow \text{Arbiter}(H, \{E_h\})$
if $c \geq \tau$ **and** $\text{Verify}(E)$ **then**
    $\mathcal{M}_t \leftarrow \text{Update}(\mathcal{M}_{t-1}, \hat{y}, E)$
else
    $\mathcal{M}_t \leftarrow \mathcal{M}_{t-1}$
end if
return $\hat{y}$

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate ERRORPROBE on three multi-agent failure attribution benchmarks (details in Appendix A). (i) **TracerTraj**: A code-generation subset of the TRACERTRAJ dataset (Zhang et al., 2026), featuring long-context traces with programmatically injected faults that provide precise, synthetic ground truth. (ii) **Who&When-Algo**: An organic failure dataset from Zhang et al. (2025b) collected from CaptainAgent, a system where agent teams are algorithmically generated and dynamically tailored to each task. (iii) **Who&When-Hand**: A complementary set of organic failures from MagenticOne (Zhang et al., 2025b), representing hand-crafted agent teams.

Compared Methods. We compare four MAS error localization methods, each instantiated with multiple backbone LLMs. **LLM-as-a-Judge** prompts a single LLM with the full trace and failure description to directly output $(\hat{a}, \hat{t})$, following prior judge-style work (Zheng et al., 2023; Liu et al., 2023; Zhang et al., 2025b); we also evaluate additional Who&When protocols (`all at once`, `step by step`, `binary search`) in Appendix B.1. **Agent-as-a-Judge (baseline)** adopts the tool-augmented framework from Zhuge et al. (2025), where the evaluator is an autonomous agent equipped with specialized capabilities (e.g., file reading, code localization, and dependency graphing) to inspect the trajectory and verify intermediate states, rather than relying solely on the static trace context. **Agent-as-a-Judge (ours)** is our full ERRORPROBE pipeline without the Verified Episodic Memory module: we first perform MAST-guided decomposition and backward tracing to compress the trace, then run the Strategist–Investigator–Arbiter team to produce $(\hat{a}, \hat{t})$. **Agent-as-a-Judge (ours, with memory)** additionally enables verified episodic memory, where tasks are processed as a stream and new patterns are committed only when verification succeeds. All four methods are compared using Claude 3.7 Sonnet, GPT-OSS-120B, and Qwen 3 32B as backbone models.[1]

Tasks and Metrics. Following our formulation in Section 3, we evaluate models on two prediction tasks: (1) **agent-level attribution**, where the system must identify the culprit agent a^*, and (2) **step-level attribution**, where it must localize the decisive error step t^*. We report top-1 *Agent* accuracy and *Step* accuracy for each dataset, and their macro-average across the three benchmarks (Table 1). Unless otherwise stated, failure modes f^* are used only as internal supervision signals rather than as an explicit evaluation target.

5.2 Main Comparison

Table 1 summarizes results across the three benchmarks.

Single-pass judging struggles with step localization on long traces. Across backbones, LLM-as-a-Judge achieves moderate agent attribution accuracy but substantially lower step-level accuracy, particularly on settings where the root cause may occur far earlier than the final symptom (e.g., TraceTraj Step $< 10\%$ for both Claude and GPT-OSS). This pattern is consistent with the intuition that a single model pass over a long trace tends to overweight late-stage symptoms and underutilize earlier causal evidence.

Tool-augmented agentic evaluation can help, but is protocol-sensitive. Agent-as-a-Judge baselines improve step-level attribution in several settings (e.g., Who&When-Algo Step increases 6.3% for Claude and 8.9% for GPT-OSS), indicating that interactive inspection can surface additional evidence beyond what the raw trace alone provides. At the same time, agent-level attribution can be brittle under this baseline protocol (e.g., Who&When-Algo Agent accuracy decreases for both Claude and GPT-OSS), motivating for a more structured diagnostic procedure.

ERRORPROBE substantially improves decisive-

[1] Many traces approach or exceed Qwen 3 32B's context window in the single-judge setting, leading to severe truncation. We therefore omit Qwen3 32B from LLM-as-a-Judge in Table 1 and only report it in Agent-as-a-Judge settings where backward tracing keeps inputs within budget.

Method	TracerTraj		Who & When (Algo)		Who & When (Hand)		Average	
	Agent	Step	Agent	Step	Agent	Step	Agent	Step
LLM-as-a-Judge								
Claude 3.7 Sonnet	67.70%	8.70%	55.60%	41.30%	43.10%	13.80%	57.03%	21.27%
GPT-OSS-120B	70.00%	7.90%	44.50%	18.50%	29.30%	19.00%	47.93%	15.13%
Agent-as-a-Judge (baseline)								
Claude 3.7 Sonnet	59.30%	11.10%	21.40%	47.60%	58.60%	15.50%	46.43%	24.73%
GPT-OSS-120B	71.10%	15.70%	18.30%	29.40%	50.00%	8.60%	46.47%	17.90%
Qwen3 32B	65.60%	12.60%	19.00%	39.70%	41.70%	15.50%	42.10%	22.60%
ERRORPROBE *(ours)*								
Claude 3.7 Sonnet	68.50%	34.60%	49.20%	58.70%	46.60%	27.6%	56.33%	41.90%
GPT-OSS-120B	70.40%	20.50%	46.80%	29.40%	56.90%	27.60%	58.46%	25.83%
Qwen3 32B	70.90%	33.10%	38.10%	28.60%	43.10%	25.90%	50.70%	29.20%
ERRORPROBE *(ours, with memory)*								
Claude 3.7 Sonnet	73.20%	39.40%	**60.30%**	**59.50%**	50.00%	29.30%	59.60%	**42.73%**
GPT-OSS-120B	71.70%	29.60%	49.20%	34.10%	**59.90%**	29.60%	**60.27%**	31.10%
Qwen3 32B	**77.80%**	**44.40%**	38.90%	29.40%	46.60%	**31.00%**	54.43%	34.93%

Table 1: Main results comparing LLM-as-a-Judge against Agent-as-a-Judge baselines and our methods. **Bold** indicates best per column. "Average" is macro-averaged to weight each benchmark equally.

step localization. Introducing MAST-guided decomposition and backward tracing (**Agent-as-a-Judge (ours)**) yields large gains in step localization across all three benchmarks for Claude and GPT-OSS. On average, ERRORPROBE improves Step accuracy from 20.6% with Claude and from 13.7% with GPT-OSS. The largest improvements occur on AgenTracer (Claude Step 25.9%; GPT-OSS Step 12.6%), consistent with ERRORPROBE's goal of recovering earlier causal steps in long trajectories rather than localizing to the final error manifestation. Agent attribution also improves substantially on Who&When-Algo (e.g., Claude Agent 27.8%; GPT-OSS Agent 28.5%), suggesting that evidence-focused tracing helps disambiguate which agent introduced the decisive mistake.

Verified memory provides additional gains, especially for weaker backbones. Enabling verified episodic memory further improves performance, with the clearest gains for GPT-OSS-120B. For Claude, memory yields smaller but consistent improvements in Step accuracy on average, with notable gains on AgenTracer (Step 4.8%) and Who&When-Algo (Step 0.8%). For Qwen3-32B in the agentic settings, memory similarly lifts both Agent and Step accuracy. Overall, these results support a key design goal of verified-before-write: memory can reuse robust diagnostic patterns while remaining controlled by explicit verification.

The strongest and most consistent benefit of ERRORPROBE is improved step-level localization under long, failure-prone trajectories. Across benchmarks, the results suggest that (i) interactive inspection is helpful but not sufficient on its own, and (ii) structured decomposition, backward causal tracing, and verification-driven memory together yield more reliable attribution than single-pass judging.

6 Analysis on Memory

6.1 In-Domain Memory Evaluation

We compare a *Baseline* (memory off) against a *Memory-Augmented* variant across four domains: MBPP and KodCode (coding), GSM8K and MATH (math reasoning). Because collecting labeled multi-agent failures is costly, we generate trajectories using multiple MAS frameworks and retain only the failed traces to focus analysis on diagnosis under error. We split traces into train and test partitions, and manually annotate test traces for evaluation.

Efficacy of in-domain memory. Table 2 shows that enabling verified memory yields consistent improvements across domains. Averaged over the four domains, memory improves agent attribution by $+10.9$ points and step localization by $+13.4$ points, indicating that diagnosis benefits from reusing previously verified error patterns in addition to per-instance reasoning.

The magnitude of improvement correlates with how repetitive error modes are within a domain.

Setting	MBPP		KodCode		GSM8K		MATH	
	Agent	Step	Agent	Step	Agent	Step	Agent	Step
ERRORPROBE *(ours)*	87.3%	33.3%	64.8%	47.2%	25.0%	30.0%	68.6%	80.0%
ERRORPROBE *(ours, with memory)*	**88.9%**	**36.5%**	**81.9%**	**56.8%**	**50.0%**	**65.0%**	68.6%	**85.7%**
Improvement	*+1.6*	*+3.2*	*+17.1*	*+9.6*	*+25.0*	*+35.0*	*+0.0*	*+5.7*

Table 2: In-domain Memory Comparison with Curated Error MAS Reasoning Traces.

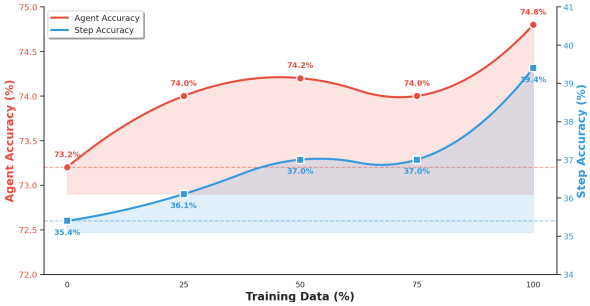

Figure 3: Learning curves for Agent and Step accuracy as memory is populated with increasing portions data.

GSM8K exhibits the largest gains (+35.0 Step), consistent with failures that often arise from repeated arithmetic or logic templates. KodCode shows strong Agent gains (+17.1), suggesting recurrent coordination and tool/API misuse patterns. In contrast, MATH sees smaller gains, consistent with more heterogeneous reasoning chains.

6.2 Memory Scaling

Figure 3 shows learning curves for Agent and Step Accuracy as memory is populated with increasing training traces from *KodCode*. Both metrics benefit from memory, but with different sensitivities. Agent Accuracy improves 1.6%, while Step Accuracy shows more pronounced gains 4.0%. This suggests base LLMs have strong priors for agent attribution but struggle with pinpointing the decisive error step; the memory module provides "error signatures" that disambiguate the root cause step.

These gains are achieved in an OOD setting, where patterns from *KodCode* are applied to *AgenTracer*. The steady upward trend indicates that the *Verified-Before-Write* gate successfully filters domain-specific noise, retaining only structural motifs that generalize across frameworks.

The steepest Step Accuracy improvement occurs between 75% and 100% data marks the diagnostic "memory" is still expanding; further scaling should yield marginal gains for rare failure modes.

6.3 Case Study: Memory Turns Abstention into Correct Attribution

In complex traces, baselines often abstain due to ambiguous evidence. Memory augmentation resolves this by retrieving verified diagnostic guards. We illustrate this in the following KodCode trace:

Scenario: The agent attempts a Shell Sort implementation.
Ground Truth: Agent=*Engineer*, Step=8.
"The Engineer claims successful Shell Sort implementation, but the file content is truncated."

Baseline Prediction:
Agent=unknown, Step=[10]
→ **Abstention** (Incorrect)

+Memory Prediction:
Agent=*Engineer*, Step=[8]
→ **Correct Attribution** (Gold step 8)

Here, the baseline abstained due to insufficient explicit evidence. However, the memory-augmented system retrieved a guard advising to "*verify that all file operations were successfully executed*". This primed the model to identify the `reasoning_action_mismatch` pattern, correctly attributing the error to unverified claims. For a detailed analysis of the failure modes resolved by memory and the specific mechanisms driving these improvements, see Appendix B.2.

7 Conclusion

We have introduced ERRORPROBE, a framework designed to resolve the challenge of semantic failure attribution in complex multi-agent workflows. By synthesizing symptom-driven backward tracing with a verified episodic memory, our approach effectively filters interaction noise and capitalizes on historical error patterns to enhance diagnostic precision. Empirical evaluations across three diverse benchmarks confirm that ERRORPROBE consistently outperforms recent baselines, specifically excelling in the difficult task of temporal error localization. Ultimately, these results underscore that structured, tool-grounded verification is a requisite component for building trustworthy multi-agent systems capable of self-improvement.

Limitations

While ERRORPROBE offers significant improvements in multi-agent failure attribution, we identify several boundaries to its current scope:

Dependency on Explicit Signals. Our symptom-driven backward tracing is predicated on the presence of detectable anomalies (e.g., exceptions, logic inconsistencies, or verifiable constraints). Consequently, "silent failures", where agents produce technically valid but semantically incorrect outputs without triggering MAST detectors, remain a challenge. This is a known hurdle in fault localization; future work may integrate test-time oracle feedbacks to expose such latent errors.

Inference Latency. The reliability of our multi-agent diagnostic team (Strategist, Investigator, Arbiter) comes at the cost of increased inference compute compared to other non-MAS baselines. While our backward tracing mechanism significantly optimizes context usage, the iterative nature of tool-assisted verification currently constrains applicability in ultra-low-latency production environments. We view this as a necessary trade-off for high-precision debugging in offline or asynchronous workflows.

Backbone Model Diversity. Due to the significant computational costs associated with multi-agent evaluation, we limited our experiments to three distinct model families. While we believe this selection provides sufficient evidence of the framework's robustness across different architectures, we were unable to exhaustively validate performance against other high-end proprietary models. Future work should aim to extend this evaluation to a broader spectrum of LLM backbones to further confirm architecture-agnostic generalizability.

Ethical Considerations

This work introduces ERRORPROBE to assist in the benign debugging of multi-agent systems; we explicitly do not endorse its use for optimizing malicious agent behaviors or circumventing safety measures. While the framework improves diagnostic precision, it relies on explicit failure signals, limiting its utility for optimizing stealthy or "silent" malicious deviations. Our experiments utilize established public benchmarks strictly adhering to their respective licenses, and do not involve the collection of new human data.

Acknowledgments

We would like to thank Thinh Vinh Ho, Kexin Wang, and Hasan Ferit Eniser from the Alexa AIDo BFA team for their valuable support with the internal experiments.

References

Adi Banerjee, Anirudh Nair, and Tarik Borogovac. 2025. Where did it all go wrong? a hierarchical look into multi-agent error attribution. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*.

Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, André F. T. Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K Surikuchi, Ece Takmaz, and Alberto Testoni. 2025. LLMs instead of human judges? a large scale empirical study across 20 NLP evaluation tasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–255.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. 2025. Who is introducing the failure? automatically attributing failures of multi-agent systems via spectrum analysis. *Preprint*, arXiv:2509.13782.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, and 15 others. 2025. Towards an ai co-scientist. *Preprint*, arXiv:2502.18864.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. 2026. Evotest: Evolutionary test-time learning for self-improving agentic systems. In *The Fourteenth International Conference on Learning Representations*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang,

Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, and 28 others. 2025. Memory in the age of AI agents. *Preprint*, arXiv:2512.13564.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*.

Jiazheng Li, Lin Gui, Yuxiang Zhou, David West, Cesare Aloisi, and Yulan He. 2023a. Distilling ChatGPT for explainable automated student answer assessment. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Jiazheng Li, Zhaoyue Sun, Bin Liang, Lin Gui, and Yulan He. 2023b. CUE: An uncertainty interpretation framework for text classifiers built on pre-trained language models. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*.

Jiazheng Li, Hainiu Xu, Zhaoyue Sun, Yuxiang Zhou, David West, Cesare Aloisi, and Yulan He. 2024. Calibrating LLMs with preference optimization on thought trees for generating rationale in science question scoring. In *Findings of the Association for Computational Linguistics: EMNLP 2024*.

Jiazheng Li, Hanqi Yan, and Yulan He. 2025a. Drift: Enhancing LLM faithfulness in rationale generation via dual-reward probabilistic inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

Jiazheng Li, Yuxiang Zhou, Junru Lu, Gladys Tyen, Lin Gui, Cesare Aloisi, and Yulan He. 2025b. Two heads are better than one: Dual-model verbal reflection at inference-time. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Xuechen Liang, Yangfan He, Yinghui Xia, Xinyuan Song, Meiling Tao, Kuan Lu, Jianhui Wang, Li Sun, Xinhang Yuan, Keqin Li, Jiaqi Chen, TIANYU SHI, and Yang Jingsong. 2026. Self-evolving agents with reflective and memory-augmented abilities. In *LLM-based Multi-Agent Systems: Towards Responsible, Reliable, and Scalable Agentic Systems*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12.

Wei Liu, Siya Qi, Yali Du, and Yulan He. 2026. Self-play only evolves when self-synthetic pipeline ensures learnable information gain. *Preprint*, arXiv:2603.02218.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522.

Xuyan Ma, Xiaofei Xie, Yawen Wang, Junjie Wang, Boyu Wu, Mingyang Li, and Qing Wang. 2025. Diagnosing failure root causes in platform-orchestrated agentic systems: Dataset, taxonomy, and benchmark. *Preprint*, arXiv:2509.23735.

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

OpenAI. 2024. Swarm.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Joseph E. Gonzalez, and Ian J. Goodfellow. 2023. MemGPT: Towards LLMs as operating systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23. ACM.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Weizhou Shen, Ziyi Yang, Chenliang Li, Zhiyuan Lu, Miao Peng, Huashan Sun, Yingcheng Shi, Shengyi Liao, Shaopeng Lai, Bo Zhang, Dayiheng Liu, Fei Huang, Jingren Zhou, and Ming Yan. 2025. QwenLong-L1.5: Post-training recipe for long-context reasoning and memory management. *Preprint*, arXiv:2512.12967.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Hanqi Yan, Linhai Zhang, Jiazheng Li, Zhenyi Shen, and Yulan He. 2025. Position: LLMs need a bayesian meta-reasoning framework for more robust and generalizable reasoning. In *Forty-second International Conference on Machine Learning Position Paper Track*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. 2025. Justice or prejudice? quantifying biases in LLM-as-a-judge. In *The Thirteenth International Conference on Learning Representations*.

Yifan Yu, Moyan Li, Shaoyuan Xu, Jinmiao Fu, Xinhai Hou, Fan Lai, and Bryan Wang. 2026. CORRECT: COndensed error RECognition via knowledge transfer in multi-agent systems.

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

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025a. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. 2025b. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *ACM Transactions on Information Systems*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36.

Yuxiang Zhou, Jiazheng Li, Yanzheng Xiang, Hanqi Yan, Lin Gui, and Yulan He. 2024. The mystery of in-context learning: A comprehensive survey on interpretation and analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-judge: Evaluate agents with agents. In *Proceedings of the 42nd International Conference on Machine Learning*.

A Experimental Details

Hyperparameters For each backbone and method, we tune any method-specific hyperparameters (e.g., number of retrieved memory entries k, verification threshold τ) on a held-out subset of TRACERTRAJ, and reuse the same configuration across all datasets to avoid test-set overfitting.

All reported numbers are averaged over 5 independent runs with default temperature. Standard deviations across runs were small (typically $\pm 1.5\%$ for Agent accuracy and $\pm 2.5\%$ for Step accuracy), so we report median only in Table 1 for clarity. Using a two-proportion z-test, the Step accuracy improvements of ERRORPROBE over LLM-as-a-Judge are statistically significant ($p<0.01$) across

all benchmarks, with effect sizes of +30.7% (TracerTraj), +18.2% (Who&When Algo), and +15.5% (Who&When Hand).

Table 3 summarizes the key hyperparameters used in ERRORPROBE. All values were tuned on a held-out subset of TRACERTRAJ (10% of traces) and fixed across all evaluation benchmarks.

Parameter	Value	Description
k	5	Number of retrieved memory entries for the Strategist
τ	0.7	Confidence threshold for memory commitment
α	0.6	Weight for semantic similarity in hybrid retrieval (vs. frequency)
d	1536	Embedding dimension for memory keys
T	0.7	Decoding temperature for LLM calls

Table 3: Hyperparameter settings for ERRORPROBE.

Benchmark Details We evaluate our framework across three distinct datasets chosen to cover a spectrum of difficulty, from simple logic errors to complex structural misalignment. Table 4 summarizes their key statistics.

TracerTraj. This benchmark is derived from the TRACERTRAJ dataset introduced by Zhang et al. (2026), specifically focusing on the code generation domain (e.g., tasks from MBPP+, KodCode). It evaluates agents within popular frameworks such as METAGPT, OPENHANDS (formerly OpenDevin), and AUTOGEN. Unlike datasets with naturally occurring errors, AGENTRACER employs a programmatic fault injection mechanism. A specific "perturbation operator" is applied to a ground-truth successful trajectory at step t (e.g., modifying a correct code block to contain a bug or altering a tool argument), thereby generating a failure trace. This methodology ensures the ground truth for failure attribution is absolute by construction, eliminating annotation ambiguity and allowing for precise evaluation of the model's ability to locate the exact "decisive error" step.

Who&When-Algo. This subset represents organic failures from the WHO&WHEN benchmark (Zhang et al., 2025b), focusing on systems with algorithmically generated team structures. The trajectories are collected from CAPTAINAGENT and its adaptive variants (e.g., AG2), where a central "Captain" dynamically orchestrates nested teams of specialized agents tailored to specific reasoning and logic queries. The errors in this dataset are non-synthetic, arising naturally during the complex interactions of these dynamically formed teams. Ground truth is established through rigorous human annotation, where experts identify the earliest action in the trace that, if corrected, would prevent the task failure. This dataset tests attribution performance in highly adaptive, fluid multi-agent environments.

Who&When-Hand. Serving as a proxy for stable, production-grade systems, this subset from Zhang et al. (2025b) features organic failures from MAGNETICONE, a high-quality, orchestrator-based system. Unlike the *Algo* subset, the agentic architecture here is fixed, involving a static set of specialized roles (e.g., WebSurfer, FileSurfer, Coder) coordinated by a central orchestrator. The tasks involve complex, multi-step real-world scenarios such as open-ended web navigation and file manipulation. Ground truth is strictly human-annotated, providing a "gold standard" for diagnosing failures in mature, tool-heavy agentic workflows where errors often stem from nuanced interaction breakdowns rather than simple logic faults.

Model Versions For reproducibility, we report the specific model versions used in our experiments. For Claude 3.7 Sonnet, we used `claude-3-7-sonnet-20250219`. For GPT-OSS-120B, we used `gpt-oss-120b-1`. For Qwen3 32B, we used `qwen3-32b-v1` via the API accessible from the Bedrock. All experiments were conducted between Oct and Dec 2025.

Computational Resources All experiments were conducted using API-based inference with no local GPU requirements. For a single diagnosis, average inference time for ERRORPROBE is approximately 45 seconds for Claude 3.7 Sonnet.

Licensing and Intended Use The evaluation benchmarks used in this work are subject to their original licenses: TracerTraj and AgenTracer are released under Apache 2.0 (Zhang et al., 2026), and Who&When is released under MIT license (Zhang et al., 2025b). We intend ERRORPROBE for research on multi-agent system debugging and do not endorse its use for circumventing safety measures or concealing malicious agent behavior.

B Further Experiments

B.1 Who&When Protocol Baselines

We also evaluate three failure attribution protocols from the Who&When benchmark (Zhang et al.,

Feature	AgenTracer	Who&When-Algo	Who&When-Hand
Source Paper	Zhang et al. (2026)	Zhang et al. (2025b)	Zhang et al. (2025b)
Size	127	127	58
Failure Origin	Synthetic	Organic	Organic
Ground Truth	Programmatic Injection	Algorithm Generated	Manual Annotation
Primary Domain	Code Generation	Logic & Reasoning	Web & Complex Tasks

Table 4: Comparison of the multi-agent failure attribution benchmarks used in evaluation.

2025b): `all at once`, `step by step`, and `binary search`. Each protocol uses an LLM to predict (i) the responsible agent and (ii) the decisive step that triggered the observed task failure. We instantiate these protocols with three representative backbones (Claude 3.7 Sonnet, GPT-OSS-120B, and Qwen3 32B) and report results on Who&When (Algorithm-Generated), Who&When (Hand-Crafted), and TracerTraj-Code. Table 5 summarizes the results.

Protocol choice induces a clear agent step trade-off. On Who&When (Algorithm-Generated), `all_at_once` achieves the strongest overall performance across models (57.94–61.11% Agent; 22.22–26.98% Step), suggesting that a single-pass judge can succeed when the trace structure is relatively regular and the failure signal is locally salient. In contrast, `step_by_step` substantially lowers agent accuracy on the same dataset (23.02–24.60% Agent) while retaining moderate step accuracy (15.08–19.05% Step), indicating that iterative decomposition can reduce reliability in agent attribution even when it remains competitive for step localization.

Hand-Crafted traces benefit from iterative refinement. On Who&When (Hand-Crafted), one-shot judging (`all_at_once`) largely fails to localize decisive steps (1.72–5.17% Step), despite moderate agent identification (34.48–41.38% Agent). Both iterative protocols improve step localization substantially, with `step_by_step` reaching up to 20.69% Step and `binary_search` reaching up to 17.24% Step. This pattern is consistent with Hand-Crafted cases requiring hypothesis refinement across multiple candidate explanations, where the earliest decisive step may be distant from the observed symptom.

TracerTraj-Code is hardest for step localization, and `binary_search` is most effective. TracerTraj-Code remains challenging for step attribution: `all_at_once` and `step_by_step` yield very low step accuracy across all models (2.36–4.72% Step). In contrast, `binary_search` improves step localization markedly (8.66–16.54% Step), while maintaining comparable agent accuracy (51.97–55.91% Agent). This suggests that for longer, code-centric traces, the decisive step is often "upstream" of the failure manifestation, and systematic narrowing over the trace is more reliable than either single-shot judgment or purely sequential inspection.

Model choice matters, but less than protocol choice for step localization. The best-performing model depends on the dataset and protocol: GPT-OSS-120B performs best on Who&When (Algorithm-Generated) under `all_at_once` (26.98% Step), while Qwen3 32B performs best on Who&When (Hand-Crafted) under `step_by_step` (20.69% Step) and on TracerTraj-Code under `binary_search` (16.54% Step). However, the variance across protocols is larger than the variance across backbones, especially on Hand-Crafted and TracerTraj-Code where the choice of attribution strategy dominates step-localization performance.

Across all settings, the reproduced Who&When protocols demonstrate that multi-agent failure attribution is sensitive not only to the judge model but also to the attribution procedure. In particular, `all_at_once` is competitive on more regular traces, while `binary_search` is consistently the most effective strategy for step localization on long and code-centric traces.

B.2 Extended Error Analysis and Mechanisms

We analyzed the 32 instances where memory augmentation corrected a previously incorrect baseline prediction. As shown in Table 6, the distribution reveals that failures related to verification protocols, specifically `incomplete_verification` (11) and `reasoning_action_mismatch` (8), are the dominant modes resolved by memory. This suggests that learned guards are particularly effective at enforcing rigorous verification protocols that baseline

Model	Who & When (Algo)		Who & When (Hand)		TracerTraj		Average	
	Agent	Step	Agent	Step	Agent	Step	Agent	Step
`all at once`								
Claude 3.7 Sonnet	60.32%	25.40%	39.66%	5.17%	57.48%	3.15%	52.49%	11.24%
GPT-OSS-120B	57.94%	26.98%	34.48%	1.72%	55.12%	2.36%	49.18%	10.35%
Qwen3 32B	61.11%	22.22%	41.38%	3.45%	53.54%	3.94%	52.01%	9.87%
`step by step`								
Claude 3.7 Sonnet	23.02%	15.08%	51.72%	17.24%	45.67%	4.72%	40.14%	12.35%
GPT-OSS-120B	24.60%	19.05%	44.83%	13.79%	45.67%	3.94%	38.37%	12.26%
Qwen3 32B	23.02%	15.87%	53.45%	20.69%	49.61%	2.36%	42.03%	12.97%
`binary search`								
Claude 3.7 Sonnet	36.51%	11.90%	43.10%	15.52%	53.54%	11.81%	44.38%	13.08%
GPT-OSS-120B	34.92%	19.84%	50.00%	13.79%	51.97%	8.66%	45.63%	14.10%
Qwen3 32B	33.33%	17.46%	53.45%	17.24%	55.91%	16.54%	47.56%	17.08%

Table 5: Who&When failure attribution baseline results using three canonical protocols: `all at once`, `step by step`, and `binary search`. We report Agent accuracy and Step accuracy on Who&When (Algo), Who&When (Hand-), and TracerTraj.

models frequently overlook.

Error Family	Count
`incomplete_verification`	11
`reasoning_action_mismatch`	8
`step_repetition`	7
`context_loss`	2
`fail_task_spec`	2
`other`	2
Total	**32**

Table 6: Distribution of error families in cases where memory enabled correct detection over the baseline.

We attribute this performance gain to three primary mechanisms facilitated by the Error Patch Memory. First, **Pattern Priming** occurs when retrieved guards prime the Analyzer to identify specific failure signatures (e.g., unverified file operations), effectively directing attention to relevant regions of the context window. Second, **Span Calibration** leverages positional metadata within memory entries to align predicted error windows to the appropriate trace segments. Finally, **Taxonomic Guidance** aids in classifying ambiguous failures into actionable categories, significantly reducing model abstention rates by providing a structured decision framework.

B.3 Illustration of Memory Content

The following box displays a representative selection of verified failure patterns stored in the episodic memory. These natural language "guards" are generated by the Investigator and validated by the Arbiter to address specific MAST failure modes, such as step_repetition or reasoning_action_mismatch. By retrieving these guards during the diagnosis of new traces, ERRORPROBE utilizes historical evidence to prime the model for specific anomalies, thereby resolving ambiguous cases where standard prompts might abstain.

Illustration of Memory Content

- Before considering a command execution complete, verify that the entire command was sent and executed successfully. For `Editor.write` commands, ensure the full file content is included and the command completes with proper JSON structure.
- After attempting to create a file, verify the file exists by using a tool like `os.path.exists()` or by listing directory contents. If file creation fails, try an alternative approach or report the error.
- Add a tracking mechanism to record what steps have already been completed. Before starting a new thinking or action step, check if it has already been done. For example: if `'task_assignment_complete'` in memory: `skip_task_assignment()`.
- Before sending a message, verify that the target agent exists in the current role list. If the specified agent is not found, raise a role-specification error or fallback to a valid agent (e.g., use the defined Engineer role instead of "Alex").
- Before issuing a finish command, the Team Leader must verify that the `solution.py` file actually exists in the workspace and that it passes all tests. Add a guard that checks the file system and runs the test suite, and only proceed to finish when both checks succeed.
- Add a guard that detects when the Engineer repeats 'thinking' steps without performing any file or code actions for more than two consecutive turns, and forces the next step to be an actionable one (e.g., create a file, write code, or run a test).
- Add a guard that detects low-complexity (XS) user requests and forces the assistant to respond directly with the solution instead of spawning additional agents.