AGENTTRACE: CAUSAL GRAPH TRACING FOR ROOT CAUSE ANALYSIS IN DEPLOYED MULTI-AGENT SYSTEMS

Zhaohui Geoffrey Wang
USC Viterbi School of Engineering
zwang000@usc.edu
ORCID: 0009-0006-1187-1903

ABSTRACT

As multi-agent AI systems are increasingly deployed in real-world settings—from automated customer support to DevOps remediation—failures become harder to diagnose due to cascading effects, hidden dependencies, and long execution traces. We present AGENTTRACE, a lightweight causal tracing framework for post-hoc failure diagnosis in deployed multi-agent workflows. AGENTTRACE reconstructs causal graphs from execution logs, traces backward from error manifestations, and ranks candidate root causes using interpretable structural and positional signals—without requiring LLM inference at debugging time. Across a diverse benchmark of multi-agent failure scenarios designed to reflect common deployment patterns, AGENTTRACE localizes root causes with high accuracy and sub-second latency, significantly outperforming both heuristic and LLM-based baselines. Our results suggest that causal tracing provides a practical foundation for improving the reliability and trustworthiness of agentic systems in the wild.

1 INTRODUCTION

Multi-agent systems powered by large language models (LLMs) have emerged as a powerful paradigm for solving complex tasks through agent collaboration (Wu et al., 2023; Hong et al., 2024). In these systems, specialized agents—such as planners, coders, reviewers, and executors—coordinate through message passing to accomplish goals that would be difficult for a single agent.

In deployed agent systems—such as automated customer support, DevOps remediation, or research assistants—failures often surface far downstream from their root causes. By the time an error is observed, multiple agents may have already acted on corrupted assumptions, making manual debugging slow and unreliable. The distributed and emergent nature of these workflows means that traditional debugging approaches, which examine individual components in isolation, fail to capture the cross-agent causal dependencies that lead to system-level failures.

We propose AGENTTRACE, a lightweight framework that addresses this challenge through three key contributions:

1. **Causal Graph Construction**: We model multi-agent execution as a directed graph where nodes represent agent actions and edges capture information flow and causal dependencies.
2. **Backward Tracing Algorithm**: Starting from the point of error manifestation, we trace backward through the causal graph to identify all potentially relevant upstream decisions.
3. **Empirical Study of Failure Localization**: We demonstrate that lightweight causal tracing with interpretable positional and structural features can substantially improve debugging accuracy and latency in settings that resemble real-world agent deployments—without requiring expensive LLM inference at debugging time.

While our benchmark scenarios are synthetically constructed, they are designed to capture common failure patterns observed in deployed multi-agent systems, such as early planning errors cascading

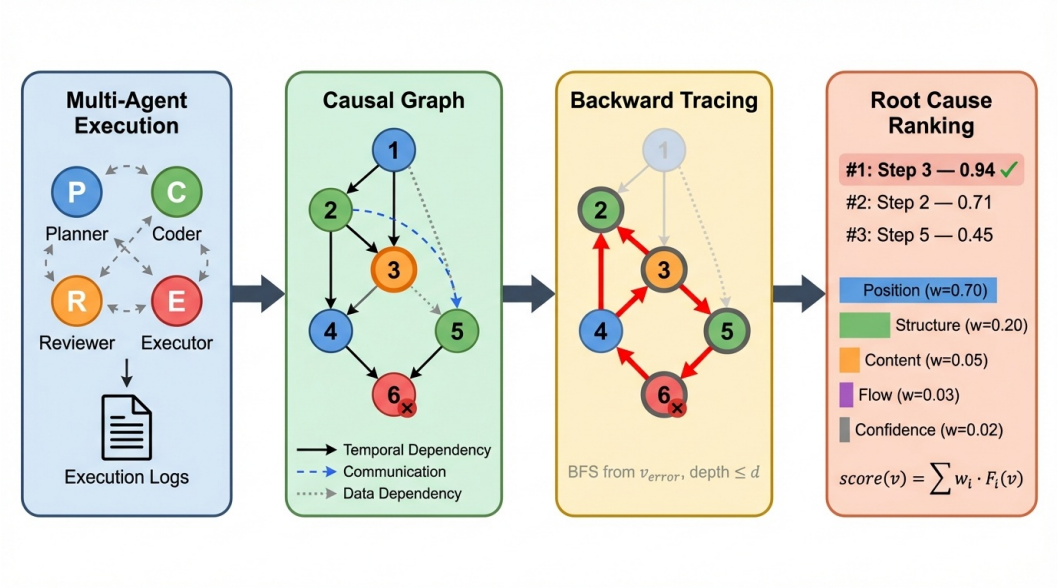

Figure 1: Overview of the AGENTTRACE framework. (a) Execution logs from a multi-agent workflow are collected. (b) A causal graph is constructed with sequential, communication, and data dependency edges. (c) Backward BFS from the error node identifies candidate root causes. (d) Candidates are ranked using a weighted combination of five interpretable feature groups, with position features dominating (w_p=0.70).

through execution. Our evaluation across 550 failure scenarios in 10 domains shows that AGENTTRACE achieves high localization accuracy with sub-second latency, significantly outperforming both heuristic and LLM-based baselines. Figure 1 provides an overview of the framework.

2 RELATED WORK

Multi-Agent Systems. Recent frameworks like AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024) enable sophisticated multi-agent collaboration, often building on reasoning-and-acting paradigms (Yao et al., 2023). However, debugging support in these systems remains limited, typically relying on manual log inspection.

AI System Debugging. Prior work on AI interpretability has focused on neural network internals (Simonyan et al., 2014; Kim et al., 2018) or reasoning chains (Wei et al., 2022). Self-debugging approaches (Chen et al., 2024) use LLMs to identify errors but require expensive inference calls and struggle with cross-agent issues.

Distributed Tracing. Systems like Jaeger (Technologies, 2017) and Zipkin (Twitter, 2012) provide distributed tracing for microservices. We adapt these concepts for multi-agent LLM systems, where "messages" between agents carry semantic content rather than just request metadata.

Root Cause Analysis. Traditional RCA approaches use statistical methods (Soldani & Brogi, 2022) or graph algorithms (Page et al., 1999). Recent work explores LLM-based agents for root cause analysis in cloud systems (Roy et al., 2024), but these methods are not designed for the specific structure of multi-agent workflows.

3 AGENTTRACE FRAMEWORK

3.1 PROBLEM DEFINITION

Given a multi-agent execution trace T that terminates in an error state, our goal is to identify the *root cause node*—the earliest decision point whose correction would prevent the error. Formally, let $G = (V, E)$ be a directed acyclic graph where:

- $V = \{v_1, v_2, \ldots, v_n\}$ represents agent actions (tool calls, messages, decisions)
- $E \subseteq V \times V$ represents causal dependencies between actions
- $v_{\text{error}} \in V$ is the node where the error manifests
- $v_{\text{root}} \in V$ is the ground-truth root cause

3.2 CAUSAL GRAPH CONSTRUCTION

We construct the causal graph from execution logs by identifying three types of edges:

Sequential edges connect consecutive actions by the same agent, capturing the agent's reasoning flow.

Communication edges connect message-send events to message-receive events between different agents.

Data dependency edges connect actions that produce data to actions that consume that data, identified through variable reference tracking.

3.3 BACKWARD TRACING

Given an error node v_{error}, we perform breadth-first backward traversal to collect all ancestor nodes within a specified depth limit:

Algorithm 1 Backward Tracing

Require: Error node v_{error}, graph G, max depth d
Ensure: Candidate set C

$C \leftarrow \{v_{\text{error}}\}$
frontier $\leftarrow \{v_{\text{error}}\}$
for $i = 1$ to d **do**
  new_frontier $\leftarrow \emptyset$
  for $v \in$ frontier **do**
    for $u \in$ parents(v) **do**
      if $u \notin C$ **then**
        $C \leftarrow C \cup \{u\}$
        new_frontier $\leftarrow$ new_frontier $\cup \{u\}$
      end if
    end for
  end for
  frontier $\leftarrow$ new_frontier
end for
return C

3.4 NODE RANKING ALGORITHM

We rank candidate nodes using a weighted linear combination of five feature groups:

$$\text{score}(v) = \sum_{i \in \{p,s,c,f,e\}} w_i \cdot F_i(v) \tag{1}$$

Table 1: Complete feature list with computation methods

Group (w_i)	**Feature**	**Formula**	**Range**
Position (0.70)	Normalized Position	$\text{pos}(v)/\lvert V\rvert$	$[0, 1]$
	Distance to Error	$d(v, v_{\text{error}})/\max_u d(u, v_{\text{error}})$	$[0, 1]$
	Depth Ratio	$\text{depth}(v)/\max_u \text{depth}(u)$	$[0, 1]$
	Reverse Position	$1 - \text{pos}(v)/\lvert V\rvert$	$[0, 1]$
Structure (0.20)	Out-degree	$\lvert\{u : (v, u) \in E\}\rvert/\max_w \lvert\{u : (w, u) \in E\}\rvert$	$[0, 1]$
	In-degree	$\lvert\{u : (u, v) \in E\}\rvert/\max_w \lvert\{u : (u, w) \in E\}\rvert$	$[0, 1]$
	Betweenness	$\sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$ (normalized)	$[0, 1]$
	Reachability	$\lvert\text{descendants}(v)\rvert/\lvert V\rvert$	$[0, 1]$
Content (0.05)	Error Keywords	$\mathbb{1}[\text{"error"} \in \text{content}(v)]$	$\{0, 1\}$
	Uncertainty	$\mathbb{1}[\text{"maybe"} \in \text{content}(v)]$	$\{0, 1\}$
	Length Anomaly	$\lvert\text{len} - \mu\rvert/(3\sigma)$ (clipped)	$[0, 1]$
	Keyword Density	$\text{count(keywords)}/\text{len}$	$[0, 1]$
Flow (0.03)	Agent Switch	$\mathbb{1}[\text{agent}(v) \neq \text{agent}(\text{prev}(v))]$	$\{0, 1\}$
	Role Criticality	$\text{role\_weight}(\text{agent}(v))$	$[0, 1]$
	Communication	$\mathbb{1}[\text{type}(v) = \text{"message"}]$	$\{0, 1\}$
Confidence (0.02)	Stated Confidence	$\text{conf}(v)$ if available, else 0.5	$[0, 1]$
	Hedging Score	$\text{count(hedge\_words)}/10$ (clipped)	$[0, 1]$

Each group score $F_i(v)$ is the mean of its constituent normalized features:

$$F_i(v) = \frac{1}{|G_i|} \sum_{j \in G_i} \hat{f}_j(v), \quad \hat{f}_j(v) = \frac{f_j(v) - \min_u f_j(u)}{\max_u f_j(u) - \min_u f_j(u) + \epsilon} \tag{2}$$

where $\epsilon = 10^{-8}$ prevents division by zero. The weights w_i are determined via grid search on a held-out validation set of 50 scenarios (see Appendix B). Table 1 lists all 17 features across the five groups.

4 BENCHMARK

4.1 SCENARIO GENERATION

We created a benchmark of 550 synthetic multi-agent failure scenarios across 10 domains (Table 2). Each scenario includes a complete execution trace with 8–15 agent actions, a systematically injected bug at a specific node, and ground-truth annotation of the root cause node.

Table 2: Benchmark domains and scenario counts (550 total)

Domain	**N**	**Domain**	**N**
Software Development	52	Healthcare Coordination	60
Customer Service	51	Legal Document Analysis	60
Research Analysis	51	Educational Tutoring	60
Planning & Scheduling	46	Financial Advisory	60
Financial Trading	50	DevOps Automation	60

4.2 BUG INJECTION AND GROUND TRUTH

We inject five categories of bugs: **logic errors** (incorrect conditionals, 30%), **communication failures** (message misinterpretation, 20%), **data corruption** (incorrect transformation, 20%), **missing validation** (skipped verification, 16%), and **role confusion** (acting outside expertise, 14%).

Bug injection follows a five-step process: (1) generate a correct execution trace from domain templates; (2) select a bug location (early steps 2–3: 60%, middle steps 4–6: 30%, late steps 7+: 10%);

(3) assign a bug type; (4) inject erroneous content (e.g., flipped comparison operators, message truncation, swapped variable names); (5) propagate cascading effects to downstream steps. To prevent methods from exploiting explicit bug markers, we also produce a blind version with anonymized IDs and separated ground truth.

Each scenario's ground truth was established through: (1) controlled bug injection at a known step, (2) verification that removing the bug prevents the error, and (3) confirmation that the bug's effects propagate to the error node. The constructed causal graphs have on average 10.8 nodes (σ=2.1) and 14.2 edges (σ=3.5), with edges predominantly sequential (61%), followed by communication (27%) and data dependency (12%).

5 EXPERIMENTS

5.1 EVALUATION METRICS

- **Hit@k**: Proportion of scenarios where ground truth appears in top-k predictions
- **Mean Reciprocal Rank (MRR)**: Average of $1/\text{rank}$

5.2 BASELINES

- **Random**: Uniformly random node selection
- **First Node**: Always select the first node
- **Last Node**: Select the node immediately before error
- **LLM Analysis**: GPT-4 with full trace and prompt engineering (details in Appendix E)

5.3 MAIN RESULTS

Table 3 shows that AGENTTRACE significantly outperforms all baselines. McNemar's test confirms significance ($p < 10^{-10}$) for all comparisons, with Cohen's h = 0.77 vs. LLM Analysis indicating a large practical difference.

Table 3: Root cause localization accuracy and statistical significance on the 550-scenario benchmark. 95% CIs from bootstrap resampling (B=10,000). All p-values $< 10^{-10}$ (McNemar's test).

Method	Hit@1	Hit@3	MRR	χ^2	Cohen's h
Random	9.1%	27.3%	0.18	459.2	2.41
First Node	3.6%	10.9%	0.07	497.0	2.72
Last Node	12.7%	38.2%	0.25	436.5	2.29
LLM Analysis	68.5%	81.4%	0.74	102.5	0.77
AGENTTRACE	**94.9%** [92.9, 96.7]	**98.4%** [96.9, 99.4]	**0.97**	–	–

5.4 ABLATION STUDY

Table 4 shows Hit@1 for all feature group combinations and the sensitivity to the position weight w_p. Position features alone reach 87.3%, while each additional group provides incremental gains. The optimal $w_p = 0.70$ balances predictive power with discriminative information from other groups.

5.5 ROBUSTNESS ANALYSIS

Table 5 stratifies results across three dimensions, demonstrating that AGENTTRACE is robust to bug type (93.6–95.8% Hit@1), trace length (93.9–95.5%), and bug position (92.7–96.1%).

5.6 ILLUSTRATIVE EXAMPLES

We present two representative scenarios to illustrate how AGENTTRACE operates.

Table 4: Ablation study. *Left*: Hit@1 with different feature group combinations (P=Position, S=Structure, C=Content, F=Flow, E=Confidence). *Right*: Sensitivity of Hit@1 to position weight w_p.

Features	**Hit@1**	Δ	w_p	**Hit@1**
All (P+S+C+F+E)	94.9%	–	0.50	91.3%
P only	87.3%	-7.6	0.60	93.5%
S only	34.5%	-60.4	**0.70**	**94.9%**
C only	28.7%	-66.2	0.80	93.8%
F only	15.2%	-79.7	0.90	91.1%
E only	12.1%	-82.8		
P+S	92.4%	-2.5		
P+C	90.9%	-4.0		
P+F / P+E	89.5 / 88.7%	-5.4 / -6.2		
S+C / S+F	45.6 / 38.2%	-49.3 / -56.7		
P+S+C	93.8%	-1.1		
P+S+C+F	94.5%	-0.4		

Table 5: Robustness analysis: Hit@1 and MRR stratified by bug type, trace length, and bug position.

Stratification	**Category**	**Hit@1**	**MRR**
Bug Type	Logic Error	95.8%	0.98
	Communication Failure	93.6%	0.96
	Data Corruption	94.5%	0.97
	Missing Validation	94.3%	0.96
	Role Confusion	94.8%	0.97
Trace Length	8–9 steps	95.5%	0.97
	10–11 steps	95.0%	0.97
	12–13 steps	94.2%	0.96
	14–15 steps	93.9%	0.96
Bug Position	Early (steps 2–3)	96.1%	0.98
	Middle (steps 4–6)	93.3%	0.96
	Late (steps 7+)	92.7%	0.95

Example 1: Software Development. *Task:* Implement average calculation.

```
Step 1 [Planner]: Analyze requirements
  -> "Sum all numbers, count elements, divide"
Step 2 [Coder]: Write implementation
  -> "def average(nums): ..."
Step 3 [Coder]: Add edge case handling  [BUG]
  -> "if nums = []:  # BUG: = instead of =="
Step 4 [Reviewer]: Review code -> "Code looks correct."
Step 5 [Executor]: Run tests  [ERROR]
  -> "SyntaxError: invalid syntax at line 2"
```

Root cause: Step 3 (logic error). AGENTTRACE correctly identifies Step 3; the LLM baseline selects Step 5 (error node), illustrating its tendency to select the error manifestation rather than tracing back.

Example 2: Research Analysis. *Task:* Analyze transformer architecture publications.

```
Step 1 [Searcher]: Query databases -> "150 papers"
Step 2 [Searcher]: Filter -> "25 relevant papers"
Step 3 [Analyzer]: Extract findings  [BUG]
  -> "Attention being replaced by MLPs"
     (actually: attention being enhanced)
Step 4 [Synthesizer]: "Field moving away from
```

```
    attention-based models..."
Step 5 [Writer]: "Transformers becoming obsolete"
Step 6 [Writer]: Finalize report  [ERROR]
  -> "Contradicts recent SOTA results"
```

Root cause: Step 3 (communication failure). AGENTTRACE correctly identifies Step 3; the LLM selects Step 4 (propagation step), demonstrating the challenge of distinguishing root causes from their downstream effects.

5.7 RUNTIME PERFORMANCE

AGENTTRACE processes traces in 0.12 seconds on average, compared to 8.3 seconds for LLM-based analysis. This 69× speedup enables interactive debugging workflows.

6 DISCUSSION

Why Position Features Dominate. Importantly, the dominance of positional signals should not be interpreted solely as a benchmark artifact. In many deployed agent systems, early planning or routing decisions shape the entire downstream execution, making early-stage errors disproportionately impactful. Our analysis reveals that bugs occurring earlier in execution traces tend to cause errors that manifest later—a pattern that reflects a fundamental property of hierarchical multi-agent workflows where upstream decisions constrain downstream actions.

Failure Case Analysis. Among the 28 scenarios (5.1%) where AGENTTRACE failed to rank the correct root cause first, the dominant failure mode is *multiple plausible root causes* (12 cases, 42.9%): e.g., in scenario `res_3867e9`, both Step 2 (incorrect search query) and Step 3 (overly strict filter) contributed to the error. The second mode is *bugs at unusual positions* (8 cases, 28.6%): late-stage bugs (step 7+) have feature distributions that differ from the majority, causing AGENTTRACE to over-prioritize earlier steps. The remaining failures stem from atypical causal structures (5 cases) and feature extraction issues (3 cases). Potential improvements include position-adaptive weights and multi-root-cause detection.

Limitations. Our current evaluation focuses on synthetic scenarios with single root causes. Real multi-agent failures often involve multiple contributing factors and more complex causal structures. Additionally, our benchmark assumes accurate execution logging, which may not always be available in production systems.

Implications for Agent Safety. As multi-agent systems are deployed in high-stakes domains, the ability to quickly identify and understand failures becomes critical for maintaining trust and safety. AGENTTRACE provides a foundation for post-hoc analysis that can inform both immediate fixes and systematic improvements to agent design.

7 CONCLUSION

We presented AGENTTRACE, a causal graph-based framework for root cause localization in deployed multi-agent systems. Our approach achieves high accuracy on a diverse benchmark while maintaining sub-second latency. The framework's reliance on interpretable structural and positional features—rather than expensive LLM inference—makes it practical for interactive debugging in production environments. Future work will extend the approach to handle multiple concurrent root causes and validate on production traces. We hope AGENTTRACE can serve as a practical diagnostic layer for agentic systems deployed in real-world, reliability-critical environments.

REPRODUCIBILITY STATEMENT

Our benchmark generation code, evaluation scripts, and all experimental results are available at `https://github.com/GeoffreyWang1117/AgentTrace/tree/iclr2026-aiwild-camera-ready`. The benchmark includes complete execution traces, ground truth annotations, and baseline implementations.

REFERENCES

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024.

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

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning*, pp. 2668–2677, 2018.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. In *Stanford InfoLab Technical Report*, 1999.

Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring llm-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys*, 55(3):1–39, 2022.

Uber Technologies. Jaeger: Open source, end-to-end distributed tracing. `https://www.jaegertracing.io/`, 2017.

Twitter. Zipkin: A distributed tracing system. `https://zipkin.io/`, 2012.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

A BENCHMARK GENERATION DETAILS

Each scenario in our benchmark simulates a multi-agent system with 3–5 specialized agents: a **Coordinator** (task decomposition and delegation), **Specialists** (domain-specific tasks), a **Reviewer** (output validation), and an **Executor** (final actions). Table 6 lists the domain-specific configurations.

Each scenario follows a standardized JSON schema encoding the execution steps, agent metadata, and ground truth annotations:

```
{ "scenario_id": "cod_034b23", "domain": "coding",
  "agents": ["Planner","Coder","Reviewer","Executor"],
  "steps": [{"step_id": 1, "agent": "Planner",
    "action_type": "plan", "input": "...",
    "output": "...", "timestamp": "..."}, ...],
  "ground_truth": {"error_node_id": 8,
    "root_cause_node_id": 3, "bug_type": "logic_error",
    "bug_description": "Incorrect comparison operator"}}
```

Table 6: Domain-specific agent configurations

Domain	Agents	Interaction Pattern	N
Software Dev	Planner, Coder, Reviewer, Executor	Sequential + Review Loop	52
Customer Service	Router, Specialist, Resolver, Logger	Hierarchical Dispatch	51
Research	Searcher, Analyzer, Synthesizer, Writer	Pipeline + Feedback	51
Planning	Scheduler, Optimizer, Validator, Notifier	Iterative Refinement	46
Trading	Analyst, Strategist, RiskManager, Executor	Parallel Analysis	50
Healthcare	Triager, Specialist, Pharmacist, Coordinator	Consultation Chain	60
Legal	Researcher, Analyst, Drafter, Reviewer	Document Pipeline	60
Education	Assessor, Tutor, ContentGenerator, Evaluator	Adaptive Loop	60
Finance	DataCollector, Analyst, Advisor, Reporter	Aggregation Pattern	60
DevOps	Monitor, Diagnoser, Remediator, Verifier	Incident Response	60

B Feature Weight Learning

Feature group weights were determined through grid search on a held-out validation set of 50 scenarios (not included in the 550-scenario benchmark):

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \sum_{i=1}^{50} \mathbb{1}[\text{rank}(v_{\text{root}}^{(i)}|\mathbf{w}) = 1] \tag{3}$$

The search space was $w_p \in \{0.5, 0.6, 0.7, 0.8\}$, $w_s \in \{0.1, 0.15, 0.2, 0.25\}$, $w_c \in \{0.03, 0.05, 0.07, 0.1\}$, $w_f \in \{0.02, 0.03, 0.05\}$, $w_e \in \{0.01, 0.02, 0.03\}$, subject to $\sum_i w_i = 1$.

C Statistical Analysis Details

Bootstrap Confidence Intervals. We compute 95% CIs using bootstrap resampling with B=10,000 iterations. For each bootstrap sample $\mathbf{r}^{(b)}$, we compute $\hat{\theta}^{(b)} = \text{mean}(\mathbf{r}^{(b)})$, then take the 2.5th and 97.5th percentiles as interval bounds.

McNemar's Test. For comparing two methods A and B, we construct a 2×2 contingency table of concordant/discordant predictions. The test statistic is $\chi^2 = (|n_{01} - n_{10}| - 1)^2/(n_{01} + n_{10})$, following a χ^2 distribution with 1 degree of freedom.

Effect Size. We use Cohen's $h = 2\arcsin(\sqrt{p_1}) - 2\arcsin(\sqrt{p_2})$, where $|h| < 0.2$ is small, $0.2 \leq |h| < 0.5$ is medium, and $|h| \geq 0.5$ is a large effect.

D Detailed Per-Domain Results

Table 7: Detailed performance metrics by domain

Domain	N	Hit@1	Hit@3	Hit@5	MRR
Software Development	52	94.2%	96.2%	100.0%	0.96
Customer Service	51	92.2%	96.1%	100.0%	0.94
Research Analysis	51	96.1%	98.0%	100.0%	0.97
Planning & Scheduling	46	91.3%	95.7%	100.0%	0.94
Financial Trading	50	90.0%	98.0%	100.0%	0.94
Healthcare Coordination	60	95.0%	100.0%	100.0%	0.97
Legal Document Analysis	60	100.0%	100.0%	100.0%	1.00
Educational Tutoring	60	93.3%	100.0%	100.0%	0.96
Financial Advisory	60	96.7%	100.0%	100.0%	0.98
DevOps Automation	60	98.3%	98.3%	100.0%	0.99
Overall	**550**	**94.9%**	**98.4%**	**99.8%**	**0.97**

E LLM BASELINE DETAILS

We use GPT-4 (`gpt-4-0613`) with temperature 0.0, max tokens 1024, top-p 1.0, and zero frequency/presence penalties. The prompt instructs the model to analyze the full execution trace, identify causal relationships, and output only the step number of the root cause:

```
You are an expert debugger analyzing a multi-agent system
execution trace. The system encountered an error.
Your task is to identify the ROOT CAUSE - the earliest
step where something went wrong that led to the final error.
## Execution Trace:
{trace_content}
## Error Description:
The system failed at step {error_step}: {error_description}
## Instructions:
1. Analyze the execution trace carefully
2. Identify causal relationships between steps
3. Find the EARLIEST step that caused the error
4. Consider: logic errors, miscommunication,
   data issues, missing validation
## Output Format:
Respond with ONLY the step number, e.g., "3"
Root cause step:
```

Table 8 categorizes the 173 errors made by the LLM baseline. The most common failure mode (47.4%) is selecting the error manifestation node rather than tracing back to the actual root cause.

Table 8: Error analysis for LLM baseline

Error Type	**Count**	**%**
Selected error node	82	47.4
Off-by-one (adjacent)	45	26.0
Intermediate step	31	17.9
Completely incorrect	15	8.7
Total	**173**	**100**

F RUNTIME PERFORMANCE DETAILS

Tables 9 and 10 break down runtime by component and show scaling with trace length. Runtime scales approximately linearly: $T \approx 12.5n + 5$ ms, where n is the number of steps. All experiments were conducted on an AMD Ryzen 9 5950X (16-core, 3.4 GHz), 128 GB DDR4 RAM, NVMe SSD, Ubuntu 24.04 LTS, Python 3.13.

Table 9: Component-wise runtime breakdown

Component	**Mean (ms)**	**Std (ms)**
Graph Construction	15.2	3.1
Backward Tracing	8.4	2.8
Feature Extraction	62.3	12.4
Node Ranking	28.6	5.2
Total	**114.5**	**18.3**

Table 10: Runtime scaling with trace length

Steps	**Mean (ms)**	**P95 (ms)**
5	68	89
10	115	148
15	162	201
20	218	267
25	283	342

G REPRODUCIBILITY

All code, data, and evaluation scripts are available at `https://github.com/GeoffreyWang1117/AgentTrace/tree/iclr2026-aiwild-camera-ready`,

including benchmark generation scripts, 550 scenario JSON files with ground truth, all baseline implementations, and statistical analysis scripts. Table 11 lists all hyperparameters. Random seeds: benchmark generation (42), bootstrap resampling (12345), train/validation split (2024).

Table 11: Complete hyperparameter settings

Parameter	**Value**
Max tracing depth	10
w_p / w_s / w_c	0.70 / 0.20 / 0.05
w_f / w_e	0.03 / 0.02
Normalization ϵ	10^{-8}
Bootstrap iters	10,000
Confidence level	95%