When Only the Final Text Survives: Implicit Execution Tracing for Multi-Agent Attribution

Yi Nian[1]* **Haosen Cao**[1]* **Shenzhe Zhu**[2] **Henry Peng Zou**[3] **Qingqing Luan**[4] **Yue Zhao**[1]
[1]University of Southern California
[2]University of Toronto
[3]University of Illinois Chicago
[4]Independent Researcher

Abstract

When a multi-agent system produces an incorrect or harmful answer, who is accountable if execution logs and agent identifiers are unavailable? Multi-agent language systems increasingly rely on structured interactions such as delegation and iterative refinement, yet the final output often obscures the underlying interaction topology and agent contributions. We introduce IET (**Implicit Execution Tracing**), a metadata-independent framework that enables token-level attribution directly from generated text and a simple mechanism for interaction topology reconstruction. During generation, agent-specific keyed signals are embedded into the token distribution, transforming the text into a self-describing execution trace detectable only with a secret key. At detection time, a transition-aware scoring method identifies agent handover points and reconstructs the interaction graph. Experiments show that IET recovers agent segments and coordination structure with high accuracy while preserving generation quality, enabling privacy-preserving auditing for multi-agent language systems.

1 Introduction

The adoption of autonomous agents is increasing rapidly; industry forecasts indicate that 40% of enterprise applications will feature task-specific AI agents by 2026 (Gartner, 2025; Zou et al., 2025; **?**). Despite this growth, recent evaluations of multi-agent frameworks report failure rates between 41% and 87% in complex tasks (Cemri et al., 2025; Miao et al., 2025). This operational opacity creates a *gap in accountability* when systems produce incorrect or harmful content.

Background. Attribution in language generation relied on explicit metadata: model identifiers, execution logs, or externally recorded provenance signals (Wang et al., 2026; Barke et al., 2026; Zhang et al., 2025c). In multi-agent language systems, this assumption becomes increasingly fragile. Current state-of-the-art diagnostic frameworks, such as Who&When (Zhang et al., 2025c) and FAMAS (Ge et al., 2025), require analyzing complete execution trajectories to perform failure attribution. These methods assume an optimal "white-box" environment where agent identities and internal logic are fully transparent. However, exposing such detailed traces introduces privacy risks and compliance constraints, especially when logs are abstracted or redacted (Xiang et al., 2024).

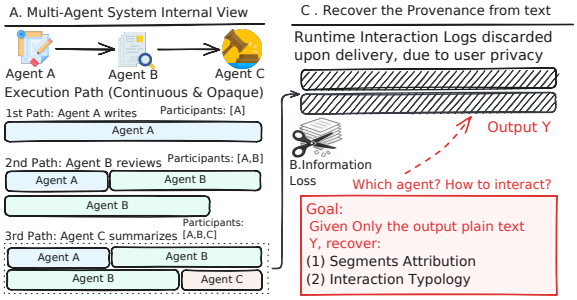

Figure 1: Agent provenance recovering problem in multi-agent systems. Internal execution paths and agent contributions are visible during generation but discarded at delivery due to privacy security, leaving only the final text. The task is to recover latent path boundaries and agent participation from text alone.

In real-world production, agent-generated content is frequently decoupled from its original environment through simple "copy-pasting" into external reports or emails. As noted by Kirchenbauer et al. (2023), once text leaves the server-side log environment, the link to its execution metadata is permanently severed. In these "out-of-band" scenarios, existing trajectory analysis tools (Zhang et al., 2025c; Ge et al., 2025) become obsolete, leaving the generated text as the **only artifact** for auditing. This necessitates an intrinsic, self-verifying mechanism that allows provenance recovery and execution tracing directly from the output, without requiring access to original system traces. In

*Equal contribution.

these scenarios, the final output becomes the **sole surviving artifact** for auditing. As LLM agents move toward high-stakes deployments, attribution can no longer depend on external metadata; instead, it requires mechanisms embedded in the generated text, turning it into key-verifiable evidence.

Furthermore, while prior work focuses predominantly on failure attribution within multi-agent system (Wang et al., 2026; Zhang et al., 2025c; Wu et al., 2025), we argue that general execution tracing is a more fundamental problem. It is essential for multi-agent accountability and credit assignment to understand "who said what" and "how they interacted",regardless of whether a failure occurred. In this sense, failure attribution can be viewed as a downstream task built upon reliable execution tracing.

Problem Definition. Multi-agent execution introduces a fundamental granularity gap between observable outputs and latent interaction structures. We formalize this as the problem of **multi-agent execution tracing**: recovering execution paths and attributing text segments to participating agents purely from the final output, without requiring explicit identity metadata or internal system traces.

Our Approach. We introduce IET or *implicit execution tracing*, a keyed, privacy-preserving attribution framework that binds execution identity directly to the token distribution during generation. Instead of storing provenance as external metadata, we embed statistically controlled, key-conditioned trace signals into the decoding process. These signals are detectable only with a secret key and remain statistically indistinguishable from natural text without it, enabling verifiable agent-level attribution. Because identity is encoded at the distributional level, attribution remains recoverable even when logs are incomplete, identifiers are removed, or orchestration details are abstracted.

At inference time, we reconstruct execution boundaries using sliding-window statistical scoring combined with change-point detection with a simple approach to record interaction patterns. This enables fine-grained segment-level attribution across diverse coordination topologies, including branching and multi-path interaction patterns.

Contributions. Our main contributions are summarized as follows:

- **New Problem Formulation.** We formalize *metadata-independent execution tracing* for multi-agent language systems, where the goal is to recover execution boundaries and attribute generated segments to participating agents using only the final output text.
- **Keyed Implicit Tracing Mechanism.** We introduce a distribution-level tracing framework that embeds agent-specific signals into the token distribution, enabling agent-level attribution without relying on explicit metadata or execution logs.
- **Boundary Reconstruction Algorithm.** We develop a sliding-window statistical scoring method combined with change-point detection to recover execution boundaries and a simple approach to record interaction paths under branching and multi-path coordination settings.
- **Empirical Evaluation.** Across diverse multi-agent coordination scenario, we demonstrate strong recoverability under identity removal and boundary perturbation while preserving generation quality.

2 Method

2.1 Problem Formulation

Given an interaction log $\mathcal{L} = (y_1, y_2, \ldots, y_T)$, our goal is to assign each token y_t to an agent $a \in \mathcal{A}$. We define this as finding an attribution function $\hat{g}(t)$ that partitions the log into segments.

Attribution via Sequential Scoring. We assume a scoring function $f(t, a)$ that evaluates the alignment between the context at time t and agent a. The attribution $\hat{g}(t)$ remains assigned to the active agent a_k until a transition to a new agent $a' \in \mathcal{A} \setminus \{a_k\}$ is identified by a detection operator D:

$$D(f, a_k, a', t, \mathcal{H}_t) > \tau. \tag{1}$$

Here, $\mathcal{H}_t$ incorporates the temporal history of scores, and τ is a sensitivity threshold. This formulation treats the interaction log as a sequence of discrete turns, where boundaries are triggered by detecting in the relative competitive scores between agents a_k and a'.

Robustness to Metadata Loss. To evaluate the restorative utility of our method, we consider an **adversarial obfuscator** $\mathcal{A}_{obf}$ that simulates metadata-independent scenarios by stripping agent identifiers and segment boundaries. Let $\Phi(G, \mathcal{L})$ be the performance of a downstream model G (e.g., error attribution) given a full-metadata log $\mathcal{L}$. Our objec-

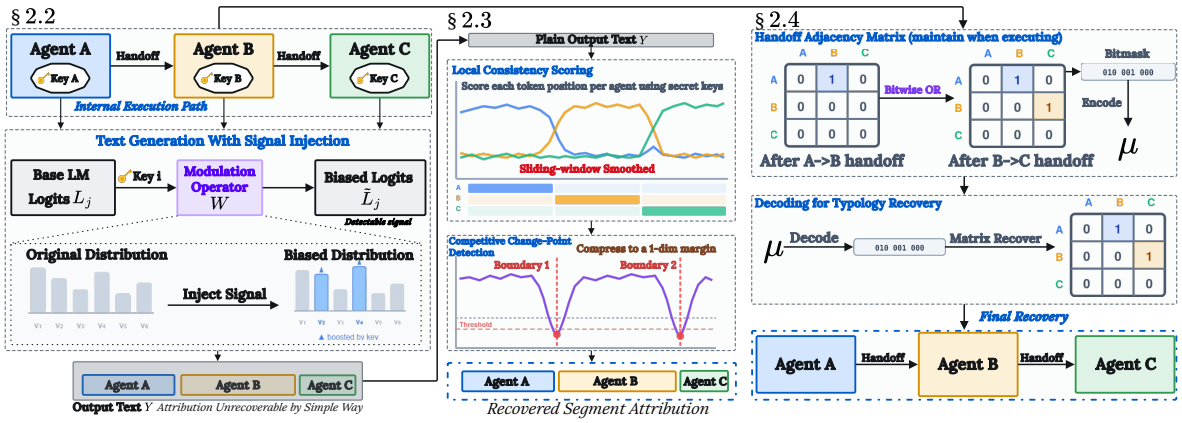

Figure 2: Overview.

tive is to ensure the *attribution consistency*:

$$\min_{\hat{g}} \ \|\Phi(G, \mathrm{Rec}(\mathcal{A}_{obf}(\mathcal{L}), \hat{g})) - \Phi(G, \mathcal{L})\| \quad (2)$$

where Rec is the reconstruction function powered by our estimated attribution $\hat{g}$. This formulation characterizes our method as a robust backbone that maintains the diagnostic power of downstream tasks even under severe information obfuscation.

Structural Validation. To validate the logical consistency of the recovered attribution, we construct an estimated interaction topology $\hat{G} = (\mathcal{A}, \hat{\mathcal{E}})$. An edge $(a^{(i)}, a^{(j)}) \in \hat{\mathcal{E}}$ is recorded for every observed transition between agent $(a^{(i)}$ and $a^{(j)}$ where i $\neq$ j. This inferred graph is then compared against the ground-truth topology $G^\star$ to evaluate the accuracy of the interaction recovery.

2.2 Implicit Execution Tracing

To enable post-hoc attribution in metadata-deprived environments, we bind each agent's identity directly to its generated output through keyed modulation of the token distribution during decoding. This process introduces agent-specific statistical signatures that can later be verified using the corresponding secret key, following the statistical signaling framework introduced by Lau et al. (2024).

Let $L_j \in \mathbb{R}^{|\mathcal{V}|}$ denote the logits produced by the base language model at token position j. For an active agent $a_k \in \mathcal{A}$ identified in the execution trace, we generate text by applying a keyed distributional modulation operator $\mathcal{W}$ conditioned on the agent identity:

$$\tilde{L}_j = \mathcal{W}(a_k, L_j). \quad (3)$$

where $\mathcal{W}$ induces a statistically detectable bias determined by a_k. Specifically, the agent identity a_k induces two distinct keys:

$$k_p = h_p(a_k), \qquad k_\pi^{(j)} = h_\pi(a_k, y_{j-n+1:j-1}), \quad (4)$$

where k_p determines a fixed perturbation direction in logit space unique to agent a_k, and $k_\pi^{(j)}$ determines a context-dependent vocabulary permutation and $y_{1:j-1}$ denotes the previously generated token sequence and $y_{j-n+1:j-1}$ represents the length-$(n-1)$ context window used to derive the context-dependent key. Similar to (Lau et al., 2024), the logits $\tilde{L}_j$ are then modulated as:

$$\tilde{L}_j = \mathcal{P}^{-1}\Big(k_\pi^{(j)},\ \mathcal{F}\big(k_p, \kappa, \mathcal{P}(k_\pi^{(j)}, L_j)\big)\Big), \quad (5)$$

where $\mathcal{P}$ handles the vocabulary permutation and $\mathcal{F}$ applies a low-magnitude perturbation κ along the direction k_p. Sampling from this modified distribution embeds the agent's identity a_k as a persistent statistical signal, enabling structural recovery even when explicit metadata is removed.

2.3 Agent Attribution

IET aims to recover the latent attribution function $\hat{g}(t)$ by identifying which agent's signal best explains each segment of the generated text Y.

Local Consistency Scoring. We first define a token-level statistic $x_j(a)$ for token at position j, to measure the alignment between the observed token y_j and the signal induced by agent $a \in \mathcal{A}$:

$$x_j(a) = \Big\langle \mathcal{P}(k_\pi^{(j)}, L_j),\ \mathcal{F}\big(k_p(a)\big) \Big\rangle, \quad (6)$$

To mitigate local variance, we instantiate the **agent-consistency metric** $f(t, a)$ from our formulation using a sliding window of width w:

$$f(t, a) = \frac{1}{w} \sum_{j=t}^{t+w-1} x_j(a). \quad (7)$$

This window-averaged score $f(t, a)$ bridges the gap between low-level token statistics and high-level structural inference.

Competitive Change-Point Detection. To identify transitions between agents, we instantiate the detection operator D using a competitive detector. We first reduce the multi-agent consistency scores into a one-dimensional competitive margin signal z_t, which quantifies the relative dominance of the leading agent candidate:

$$z_t = f(t, \hat{a}_t) - \max_{a' \in \mathcal{A} \setminus \{\hat{a}_t\}} f(t, a'), \tag{8}$$

where $\hat{a}_t = \arg\max_{a \in \mathcal{A}} f(t, a)$ is the agent with the highest alignment score at time t.

To robustly localize transitions, we smooth the margin sequence with cumulative sum algorithm to $\{z_t\}$. For the k-th segment starting at position $\hat{\tau}_{k-1}$, we track the cumulative deviation from the local mean margin $\bar{z}_k$:

$$C_t = \sum_{i=\hat{\tau}_{k-1}}^{t} \left(z_i - \bar{z}_k\right). \tag{9}$$

We formally define the detection operator D as a binary decision at the potential change point $\hat{\tau}_k = \arg\max_{t > \hat{\tau}_{k-1}} |C_t|$:

$$D(f, a', \hat{a}_t, \hat{\tau}_k) = \mathbb{I}\left(|C_{\hat{\tau}_k}| > \tau\right), \tag{10}$$

where τ is the sensitivity threshold. By iteratively identifying these extrema, we partition the interaction log into K contiguous segments. This induced segmentation allows us to recover the **attribution function** $\hat{g}(t)$ as a piecewise-constant mapping:

$$\hat{g}(t) = \hat{a}_k, \quad \text{for } t \in [\hat{\tau}_{k-1}, \hat{\tau}_k), \tag{11}$$

where $\hat{a}_k$ is the dominant agent identified within the k-th segment.

2.4 Execution Path and Topology Encoding

To recover the interaction topology $\hat{G}$, we must encode the directional dependencies between agents within the agent signal. We represent this evolving state as an online-updated adjacency matrix.

Topology State Representation. Let $\mathcal{A} = \{a_1, \ldots, a_K\}$ be the set of agents. At any step t, the execution path is represented by a binary adjacency matrix $\mathbf{M}_t \in \{0, 1\}^{K \times K}$, where $(\mathbf{M}_t)_{i,j} = 1$ if a directed interaction from agent a_i to agent a_j has occurred (e.g., through state-passing, rewriting, or observation). The path identifier μ_t is derived by flattening and encoding $\mathbf{M}_t$ as a bitmask:

$$\mu_t = g(\mathbf{M}_t) = \sum_{i=1}^{K} \sum_{j=1}^{K} (\mathbf{M}_t)_{i,j} \cdot 2^{(i-1)K + (j-1)}. \tag{12}$$

This bijective mapping ensures that μ_t uniquely identifies the current interaction graph, distinguishing between fundamental topologies such as chains, stars, or cycles.

Interactive State Maintenance. The matrix $\mathbf{M}_t$ is maintained across agent invocations to capture the cumulative context. Initialized at $\mathbf{M}_0 = \mathbf{0}$, whenever agent a_i transmits information to agent a_j, the state is updated via a bitwise OR operation:

$$\mathbf{M}_t = \mathbf{M}_{t-1} \vee \mathbf{E}_{i,j}, \tag{13}$$

where $\mathbf{E}_{i,j}$ is a sparse matrix with a 1 at index (i, j). This monotonic update rule ensures that once a dependency is established, it is persistently encoded into the subsequent watermarked tokens. By detecting the latent μ_t from the output text, we can invert the mapping to reconstruct the full adjacency matrix, thereby recovering the interaction topology $\hat{G}$ without explicit metadata.

3 Experimental Setting

3.1 Datasets

We evaluate our framework on two complementary multi-agent benchmarks that provide structured interaction logs and ground-truth speaker annotations. Additional details are provided in Appendix B.

Multi-Agent Interaction Dataset (Liu et al., 2025). This dataset contains structured multi-agent dialogues under diverse coordination paradigms, including hierarchical delegation and collaborative planning. Each interaction log provides explicit agent identities and turn-level boundaries, allowing analysis of agent participation and interaction structure. Further details of the MAMA topology dataset, including the source file and its fields, are provided in Appendix B.4.

Who & When (Zhang et al., 2025c). This benchmark focuses on speaker and error attribution in multi-agent transcripts. The dataset contains conversational logs with annotated speaker identities and temporal ordering, enabling evaluation of attribution in multi-party interactions. Additional details of the Who&When benchmark are provided in Appendix B.5.

3.2 Experimental Design

To evaluate the capabilities defined in Section 2.1, we design two complementary experimental settings that test (i) topology-sensitive attribution under controlled interaction structures and (ii) robustness of attribution under metadata obfuscation.

Topology-aware attribution. We evaluate attribution and structural recovery using the Multi-Agent Interaction Dataset (Liu et al., 2025). This dataset provides structured interaction logs where multiple agents collaborate through predefined communication graphs. To analyze how interaction topology affects attribution difficulty, we consider several canonical coordination structures, including *star*, *ring*, and *tree* topologies. For each topology, agents interact through a fixed communication protocol while solving the same set of tasks. The resulting logs allow us to examine whether the recovered attribution function can correctly identify agent contributions and reconstruct the underlying interaction topology.

Robust attribution under metadata loss. We evaluate robustness using the Who&When benchmark (Zhang et al., 2025c), which contains multi-agent transcripts with annotated speaker identities and error attribution labels. To simulate metadata-independent scenarios and verify IET framework's robustness, we apply two types of transcript perturbations: (i) **ID Removal**, where explicit agent identifiers are removed from each utterance, and (ii) **Boundary Corruption**, where utterance boundaries are shuffled or partially merged. These perturbations mimic realistic situations where execution metadata is unavailable or unreliable. The goal is to evaluate whether agent identities and interaction structures can be recovered from text alone under increasing levels of transcript corruption.

3.3 Evaluation Metrics

Following the problem formulation in Section 2.1, we evaluate three aspects of execution tracing: sequential attribution, structural validation, and robustness under metadata loss.

Sequential Attribution. We measure whether the recovered attribution function $\hat{g}(t)$ correctly assigns tokens to generating agents. Given predicted token segments $\hat{S}$ and ground-truth segments S, we report token-level accuracy and concatenated-text IoU:

$$\text{TokenAcc} = \frac{1}{T}\sum_{t=1}^{T}\mathbb{I}[\hat{g}(t) = g(t)] \tag{14}$$

$$\text{IoU}(S, \hat{S}) = \frac{|S \cap \hat{S}|}{|S \cup \hat{S}|} \tag{15}$$

We report both mean and median IoU across logs to account for variability in interaction length.

Structural Validation. To evaluate whether the recovered attribution preserves interaction structure, we construct an interaction graph from predicted agent transitions and compare it with the ground-truth topology. We report topology attribution accuracy:

$$\text{TopoAcc} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[\hat{a}_i = a_i], \tag{16}$$

where a_i and $\hat{a}_i$ denote the ground-truth and predicted agent for interaction unit i.

Robustness under Metadata Loss. To evaluate robustness under transcript corruption, we follow the evaluation protocol of the Who&When benchmark (Zhang et al., 2025c); additional benchmark details are provided in Appendix B.5. Each failure trajectory provides annotations identifying the failure-responsible agent $a^\star$ and the decisive error step $t^\star$. We simulate metadata-independent scenarios by applying two perturbations: (i) **ID Removal** and (ii) **Boundary Corruption** Given the corrupted transcript, our method first reconstructs agent identities and interaction boundaries, producing a recovered execution log. Failure attribution is then performed as a downstream task on the reconstructed structure using the Who&When evaluation protocol. Given predictions $(\hat{a}, \hat{t})$, we report:

$$\text{AgentAcc} = \mathbb{I}[\hat{a} = a^\star], \qquad \text{StepAcc} = \mathbb{I}[\hat{t} = t^\star]. \tag{17}$$

AgentAcc measures whether the correct failure-responsible agent is identified, while StepAcc evaluates localization of the decisive error step. This setup treats failure attribution as a downstream task and measures whether reconstructed execution traces preserve the diagnostic utility of the original logs.

3.4 Baselines

We compare our method against two categories of baselines: LLM-based attribution baselines and traditional segmentation baselines.

For the LLM-based baselines, we use **ChatGPT-4o** (OpenAI et al., 2023) and **DeepSeek-v3.1**

Method	Metric	#Agents = 4			#Agents = 5			#Agents = 6		
		Star-Pure	*Chain*	*Tree*	*Star-Pure*	*Chain*	*Tree*	*Star-Pure*	*Chain*	*Tree*
LLM Baselines										
ChatGPT	IoU	0.166	0.168	0.171	0.139	0.142	0.146	0.099	0.105	0.103
	TokenAcc	0.280	0.283	0.287	0.232	0.241	0.238	0.177	0.188	0.183
DeepSeek	IoU	0.175	0.165	0.178	0.149	0.140	0.156	0.111	0.111	0.125
	TokenAcc	0.292	0.279	0.297	0.251	0.244	0.263	0.198	0.198	0.219
Segmentation Baselines										
Random (token)	IoU	0.387	0.397	0.423	0.343	0.356	0.376	0.306	0.333	0.313
	TokenAcc	0.555	0.554	0.582	0.494	0.518	0.527	0.453	0.477	0.450
Random-Unit	IoU	0.424	0.435	0.420	0.353	0.370	0.398	0.316	0.347	0.333
	TokenAcc	0.583	0.589	0.573	0.525	0.524	0.507	0.460	0.469	0.459
Recursive	IoU	0.556	0.451	0.529	0.489	0.385	0.483	0.430	0.306	0.458
	TokenAcc	0.706	0.606	0.686	0.652	0.546	0.642	0.608	0.448	0.632
Semantic-BoW	IoU	0.305	0.346	0.329	0.267	0.315	0.304	0.247	0.262	0.281
	TokenAcc	0.457	0.517	0.506	0.442	0.465	0.469	0.354	0.380	0.409
TextTiling	IoU	0.390	0.414	0.402	0.364	0.335	0.397	0.347	0.291	0.374
	TokenAcc	0.513	0.568	0.540	0.474	0.498	0.471	0.428	0.406	0.468
Our Method										
Ours (WM)	IoU ↑	**0.934**	**0.938**	**0.936**	**0.932**	**0.939**	**0.933**	**0.929**	**0.939**	**0.932**
	TokenAcc ↑	**0.951**	**0.954**	**0.957**	**0.948**	**0.950**	**0.953**	**0.9417**	**0.944**	**0.949**
	TopoAcc ↑	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 1: Sequential attribution performance across LLM baselines, segmentation baselines, and our method under different agent counts and coordination topologies.

(DeepSeek-AI, 2024). Given LLM the number of agents K, we provide the LLM with the agent interaction logs and ask it to assign a span to one of the K speakers. Concretely, we first keep only assistant outputs, reconstruct the conversation in a fixed order, and split the concatenated text into fixed-length token units. The LLM then predicts a set of token-span ranges for each speaker, from which we derive token-level speaker labels. This setting requires the model to jointly infer both segmentation boundaries and speaker ownership from text alone. The full system and user prompt templates for the LLM baseline are provided in Appendix A.1; a brief overview of the prompting design is given in Appendix A.1.1.

Traditional Baselines. We compare against several segmentation-based baselines that do not model speaker identity and therefore can only predict segment boundaries. The baselines include: (i) **Random (token)**: boundaries are sampled uniformly from the token sequence. (ii) **Random-Unit**: boundaries are sampled only from unit boundaries (e.g., sentence or predefined units) rather than arbitrary token positions. (iii) **Recursive**: the text is recursively partitioned using simple textual separators and then adjusted to match the target number of segments. (iv) **Semantic-BoW**: boundaries are detected by identifying drops in bag-of-words similarity between neighboring text windows. (v) **TextTiling**: a lexical-cohesion method that places boundaries according to depth scores derived from local similarity changes across adjacent text blocks.

Since these baselines cannot infer speaker identity, we evaluate them only on the segmentation component of the task. Each method predicts $K-1$ boundaries, producing K text segments. We then align the predicted segments with the gold segments by segment index and measure segmentation quality using concatenated-text IoU and token-level accuracy. This protocol evaluates the traditional baselines on the subproblem they are designed to handle (boundary detection), while the LLM-based baselines are evaluated on the full attribution task.

3.5 Evaluation

Our evaluation is organized around three questions that correspond to the core objectives of metadata-independent execution tracing: recovering token-level attribution, reconstructing interaction structure, and preserving downstream diagnostic utility under metadata loss.

How accurately can sequential attribution be recovered? We report TokenAcc and IoU across different agent counts and interaction topologies in Table 1. LLM-based baselines (ChatGPT and

DeepSeek) perform poorly, with TokenAcc below 0.30 and IoU around 0.10–0.18. Structure-based segmentation methods improve performance (e.g., Recursive reaching IoU up to 0.56), but remain far below reliable attribution. In contrast, our method consistently achieves over 94% TokenAcc and around 0.93 IoU across all settings, while also recovering the interaction topology perfectly.

How well does the recovered trace preserve interaction structure? Table 1 also reports topology recovery accuracy (TopoAcc). Our method achieves near-perfect structural recovery across all configurations, with TopoAcc equal to 1.0 in almost all cases and never dropping below 0.98. This is because the tracing mechanism maintains an explicit execution state during generation, enabling reliable reconstruction of agent transitions and interaction topology.

Can the recovered trace support downstream failure attribution under metadata loss? Table 2 reports failure attribution accuracy under metadata corruption. Without trace signals, baseline methods degrade sharply once metadata is removed. In particular, AgentAcc drops to near zero under ID removal for both Step-by-Step and Binary Search. In contrast, our tracing-enabled method remains substantially more robust across all corruption settings. For example, under ID removal the Step-by-Step strategy still achieves 23.81% AgentAcc and 15.24% StepAcc, while Binary Search maintains non-trivial recovery performance. These results show that the recovered traces preserve sufficient structural information to support downstream failure attribution when metadata is unavailable.

Case Study: Robustness to PII Redaction. We further examine whether privacy-preserving preprocessing affects trace recovery. In the Multi-Agent Interaction Dataset, conversation logs may contain sensitive information such as names, emails, or phone numbers. To simulate realistic deployment settings, we apply standard PII redaction that replaces sensitive spans with placeholder tokens (e.g., [NAME], [EMAIL]).

Figure 3 compares tracing performance before and after redaction across different multi-agent interaction structures. Despite removing lexical content, the overall performance remains nearly unchanged. For most interaction patterns (e.g., 4-chain, 4-tree, and 5-chain), the IoU and token-level accuracy curves for the original and censored logs almost overlap. Minor deviations only appear in more complex structures such as 6-tree and 6-star-pure, where the performance drop remains small.

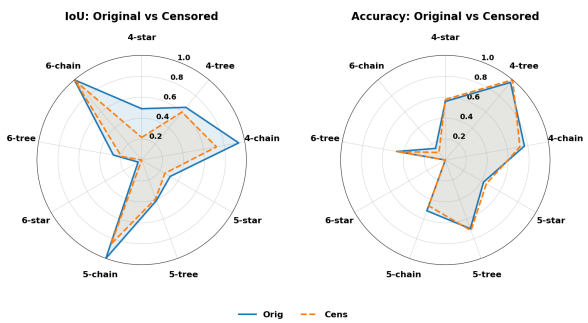

Figure 3: Robustness to PII redaction. Attribution performance on original and PII-redacted transcripts across different interaction structures. The similar curves indicate minimal degradation after redaction.

These results indicate that the tracing signal is embedded in the generation process rather than relying on surface lexical patterns. Consequently, removing sensitive tokens does not significantly affect agent attribution, demonstrating that the proposed tracing mechanism is robust to privacy-preserving transcript sanitization.

4 Related Work

4.1 Failure Attribution in Multi-Agent Systems

As large language models are increasingly deployed in multi-agent settings, understanding and debugging system failures has become an important research problem (Chen et al., 2024; Wang et al., 2026; **?**; **?**). Prior work primarily studies *failure attribution* as a post-hoc inference task: given an execution trace produced by multiple interacting agents, the goal is to infer which agent, or which step in the interaction, is responsible for a system-level failure (Zhang et al., 2025c; Ge et al., 2025; Barke et al., 2026; Cemri et al., 2025).

Recent benchmarks formalize this problem explicitly by asking models to predict the responsible agent (*who*) and the decisive error step (*when*) from raw execution traces. Other approaches employ counterfactual replay, causal analysis (Zhang et al., 2025a), or learned tracer models (Zhu et al., 2025; Zhang et al., 2025b) to identify failure sources in the absence of explicit provenance. These methods highlight the difficulty of attribution when execution logs record only surface-level interactions without preserving information-flow dependencies. Our approach is complementary: instead of relying solely on retrospective inference, we embed trace-

Method	Baseline (Zhang et al., 2025c)						Ours					
	Baseline		Remove ID		Boundary		Baseline		Remove ID		Boundary	
	Agent	*Step*	*Agent*	*Step*	*Agent*	*Step*	*Agent*	*Step*	*Agent*	*Step*	*Agent*	*Step*
All-at-Once	54.33	12.50	5.22	12.17	25.49	15.69	41.46	9.76	26.47	11.76	38.71	6.45
Step-by-Step	35.20	25.51	0.00	10.28	36.04	14.41	28.70	12.04	23.81	15.24	27.00	9.00
Binary Search	44.13	23.98	0.00	3.17	32.54	1.59	34.13	1.59	22.22	2.38	36.51	2.38

Table 2: Failure attribution accuracy under metadata corruption. Left: baseline methods without trace signals. Right: our tracing-enabled method. *Agent* and *Step* denote agent-level and step-level attribution accuracy (%).

ability into the generation process so that agent-level provenance can be recovered more directly.

4.2 Watermarking and Provenance in Large Language Models

Watermarking has been widely studied for identifying AI-generated content and verifying model provenance (Fang et al., 2025). Existing methods embed detectable signals at different stages of the generation pipeline, including during logits generation (Kirchenbauer et al., 2023; Lee et al., 2024; Hu et al., 2023; Wu et al., 2024), token sampling (Christ et al., 2024; Kuditipudi et al., 2024; Hou et al., 2024), or model training (Sun et al., 2022, 2023; Gu et al., 2023). These approaches primarily focus on detecting whether a piece of text was generated by a particular model, and are typically designed for single-model settings rather than tracing information flow across multiple agents.

A separate line of work studies watermarking for existing text through format-based, lexical-based, syntactic-based, and generation-based transformations (Brassil et al., 1995; Topkara et al., 2006; Atallah et al., 2001; Abdelnabi and Fritz, 2021). However, these approaches embed signals into a single text instance and do not model how provenance propagates across multiple interacting agents or intermediate steps. In contrast, we repurpose watermarking as a *system-level instrumentation mechanism* that preserves and propagates provenance across agent interactions, enabling explicit and verifiable attribution in multi-agent systems.

4.3 Inference versus Instrumentation

Accountability in multi-agent systems can be approached in two ways: *inference* and *instrumentation*. Inference-based approaches seek to determine responsibility after a failure occurs by analyzing execution traces that lack explicit provenance signals. In contrast, instrumentation embeds traceability directly into the system during generation, so that the origin of each contribution can be recovered explicitly. Prior work on multi-agent failure attribution largely follows the inference paradigm, while watermarking embeds identifiable signals but is typically designed for single-model settings. Our work bridges these two paradigms by repurposing watermarking as a system-level instrumentation mechanism. By embedding watermark-based provenance into multi-agent interactions, responsibility can be recovered deterministically and audited by design.

5 Conclusion

In this paper, we introduced Implicit Execution Tracing, a framework that addresses the structural opacity of modern multi-agent language systems. By embedding key-conditioned trace signals directly into the token distribution, the generated text becomes a **self-describing execution record**, enabling fine-grained agent attribution and reconstruction of interaction structures directly from the output, without relying on explicit execution logs.

Our results show that reliable tracing remains possible even in metadata-free settings where internal trajectories are unavailable for privacy or compliance reasons. As multi-agent systems evolve into complex multi-vendor ecosystems, decoupling provenance from system infrastructure provides a practical path toward accountable and privacy-preserving AI collaboration.

6 Ethical Considerations

Our research focuses on attribution and execution tracing in multi-agent language systems. To systematically evaluate our approach, we conduct experiments on conversational datasets that may contain personally identifiable information (PII), such as names or contact details. These datasets are obtained from publicly available resources and are used solely for research purposes in a controlled experimental setting. To mitigate privacy risks, we apply standard PII redaction procedures by replacing sensitive spans with placeholder tokens (e.g., [NAME], [EMAIL]). Our work aims to improve

transparency and accountability in multi-agent AI systems by enabling reliable attribution of generated content, which aligns with responsible AI development and auditing practices.

7 Limitations

Our research focuses on attribution and execution tracing in multi-agent language systems. To evaluate our approach, we conduct experiments on conversational datasets that may contain personally identifiable information (PII), such as names or contact details. These datasets are obtained from publicly available resources and used solely for research purposes in a controlled setting. To mitigate privacy risks, we apply standard PII redaction by replacing sensitive spans with placeholder tokens. Our work aims to improve transparency and accountability in multi-agent AI systems.

References

Sahar Abdelnabi and Mario Fritz. 2021. Adversarial watermarking transformer: Towards tracing text provenance with data hiding. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 121–140. IEEE.

Mikhail J. Atallah, Victor Raskin, Michael Crogan, Christian Hempelmann, Florian Kerschbaum, Dina Mohamed, and Sanket Naik. 2001. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Information Hiding*, volume 2137 of *Lecture Notes in Computer Science*, pages 185–200. Springer.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. 2026. AgentRx: Diagnosing AI agent failures from execution trajectories. *Preprint*, arXiv:2602.02475.

Jack T. Brassil, Steven Low, Nicholas F. Maxemchuk, and Lawrence O'Gorman. 1995. Electronic marking and identification techniques to discourage document copying. *IEEE Journal on Selected Areas in Communications*, 13(8):1495–1504.

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 *Advances in Neural Information Processing Systems 38: Datasets and Benchmarks Track*. Spotlight.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, and 1 others. 2024. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Miranda Christ, Sam Gunn, and Or Zamir. 2024. Undetectable watermarks for language models. In *Proceedings of the 37th Conference on Learning Theory*, volume 247 of *Proceedings of Machine Learning Research*, pages 1125–1139. PMLR.

DeepSeek-AI. 2024. DeepSeek-V3 technical report. *Preprint*, arXiv:2412.19437.

Liancheng Fang, Aiwei Liu, Henry Peng Zou, Yankai Chen, Hengrui Zhang, Zhongfen Deng, and Philip S. Yu. 2025. MUSE: Model-agnostic tabular watermarking via multi-sample selection. *Preprint*, arXiv:2505.24267.

Gartner. 2025. Gartner predicts 40% of enterprise apps will feature task-specific AI agents by 2026, up from less than 5% in 2025. Press release, updated September 5, 2025.

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.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. 2023. On the learnability of watermarks for language models. *Preprint*, arXiv:2312.04469.

Abe Bohan Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. 2024. SemStamp: A semantic watermark with paraphrastic robustness for text generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4067–4082. Association for Computational Linguistics.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. 2023. Unbiased watermark for large language models. *Preprint*, arXiv:2310.10669.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2024. Robust distortion-free watermarks for language models. *Transactions on Machine Learning Research*.

Gregory Kang Ruey Lau, Xinyuan Niu, Hieu Dao, Jiangwei Chen, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. 2024. Waterfall: Framework for robust and scalable text watermarking and provenance for LLMs. *Preprint*, arXiv:2407.04411.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2024. Who wrote this code? watermarking for code generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4890–4911. Association for Computational Linguistics.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252. Association for Computational Linguistics.

Jinbo Liu, Defu Cao, Yifei Wei, Tianyao Su, Yuan Liang, Yushun Dong, Yan Liu, Yue Zhao, and Xiyang Hu. 2025. Topology matters: Measuring memory leakage in multi-agent LLMs. *Preprint*, arXiv:2512.04668.

Chunyu Miao, Henry Peng Zou, Yangning Li, Yankai Chen, Yibo Wang, Fangxin Wang, Yifan Li, Wooseong Yang, Bowei He, Xinni Zhang, and 1 others. 2025. RECODE-H: A benchmark for research code development with interactive human feedback. *Preprint*, arXiv:2510.06186.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, and 1 others. 2023. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Zhensu Sun, Xiaoning Du, Fu Song, and Li Li. 2023. CodeMark: Imperceptible watermarking for code datasets against neural code completion models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1561–1572. ACM.

Zhensu Sun, Xiaoning Du, Fu Song, Mingze Ni, and Li Li. 2022. CoProtector: Protect open-source code against unauthorized training usage with data poisoning. In *Proceedings of the ACM Web Conference 2022*, pages 652–660. ACM.

Umut Topkara, Mercan Topkara, and Mikhail J. Atallah. 2006. The hiding virtues of ambiguity: Quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of the 8th Workshop on Multimedia and Security*, pages 164–174. ACM.

Junjie Wang, Yawen Wang, Mengzhuo Chen, Xiaofei Xie, Chunyang Chen, Fangwen Mu, Zhe Liu, and Qing Wang. 2026. A survey for LLM agent trajectory analysis: From failure attribution to enhancement. Preprint on ResearchGate.

Yaozu Wu, Dongyuan Li, Yankai Chen, Renhe Jiang, Henry Peng Zou, Wei-Chieh Huang, Yangning Li, Liancheng Fang, Zhen Wang, and Philip S. Yu. 2025. Multi-agent autonomous driving systems with large language models: A survey of recent advances. *Preprint*, arXiv:2502.16804.

Yihan Wu, Zhengmian Hu, Hongyang Zhang, and Heng Huang. 2024. DiPmark: A stealthy, efficient and

resilient watermark for large language models. Withdrawn submission to ICLR 2024.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. 2024. GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning. *Preprint*, arXiv:2406.09187.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380. Association for Computational Linguistics.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. 2025a. AgenTracer: Who is inducing failure in the LLM agentic systems? *Preprint*, arXiv:2509.03312.

Heng Zhang, Yuling Shi, Xiaodong Gu, Haochen You, Zijian Zhang, Lubin Gan, Yilei Yuan, and Jin Huang. 2025b. GraphTracer: Graph-guided failure tracing in LLM agents for robust multi-turn deep search. *Preprint*, arXiv:2510.10581.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. 2025c. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 76583–76599. PMLR.

Chenyang Zhu, Spencer Hong, Jingyu Wu, Kushal Chawla, Yuhui Tang, Youbing Yin, Nathan Wolfe, Erin Babinsky, and Daben Liu. 2025. RAFFLES: Reasoning-based attribution of faults for LLM systems. Poster at MTI-LLM @ NeurIPS 2025.

Henry Peng Zou, Wei-Chieh Huang, Yaozu Wu, Yankai Chen, Chunyu Miao, Hoang Nguyen, Yue Zhou, Weizhi Zhang, Liancheng Fang, Langzhou He, Yangning Li, Dongyuan Li, Renhe Jiang, Xue Liu, and Philip S. Yu. 2025. LLM-based human-agent collaboration and interaction systems: A survey. *Preprint*, arXiv:2505.00753.

A Appendix

A Prompt Details

This section presents the full prompt templates used in our experiments. We organize them into two groups: (1) prompts used in the LLM baseline for speaker-range assignment, and (2) prompts used in the debate framework, including the debating agents and the final judge. Placeholders enclosed in braces (e.g., {question_text}, {choices_block}) denote instance-specific fields filled dynamically at inference time.

A.1 LLM Baseline Prompts

This subsection presents the prompts used in the LLM baseline for speaker-range assignment on concatenated multi-speaker traces. Given a tokenized trace containing utterances from exactly K speakers, the model is asked to assign each contiguous token unit to one speaker and return the final segmentation in JSON format.

A.1.1 Baseline Overview

The baseline prompt consists of two parts: a system prompt and a user prompt. The system prompt specifies the output schema and the global validity constraints, including complete coverage, no overlap, boundary alignment, and non-empty assignment for every speaker. The user prompt provides the instance-specific information, including the number of speakers, the number of units, the total number of tokens, the valid unit boundaries, the task description, and the unit-level trace content.

Together, these prompts cast the attribution task as a constrained structured prediction problem: the model must first assign each unit to exactly one speaker and then merge adjacent units belonging to the same speaker into minimal non-overlapping ranges. The full prompt templates are provided below.

LLM Baseline Prompt (System Prompt). *System prompt template used in the LLM baseline for speaker-range assignment.*

```
You assign each token span in a concatenated
↪  multi-speaker trace to one of K speakers.
You must reason privately and NEVER reveal your
↪  reasoning.
Your final answer must be EXACTLY ONE JSON object and
↪  NOTHING ELSE.
Do not output any natural language, analysis, prefacing
↪  text, markdown, code fences, bullets, or apologies.
Do not start with phrases like 'Okay', 'Let's', 'Here
↪  is', or any sentence.
The ONLY valid output format is:
{"ranges": {"0": [[0, 64]], "1": [[64, 128]], "2": [[128,
↪  192]], "3": [[192, 256]]}}
Example valid output:
```

Dataset	Subset / File	Source	Role in Experiments
C4	`realnewslike`	`allenai/c4`	Initial waterfall watermark testing
TruthfulQA	multiple-choice	TruthfulQA	Debate evaluation
HotpotQA	`distractor`, `train`	`hotpot_qa`	Sliding-window and concatenated-text watermark experiments
MAMA topology	`llama3.1_num484_nop`	MAMA repository	Topology-based multi-agent experiments
Who & When	`Algorithm-Generated`, `Hand-Crafted`	Agents Failure Attribution repository	Failure attribution under metadata corruption

Table 3: Datasets used in different stages of our experiments.

```
{"ranges": {"0": [[0, 64]], "1": [[64, 128], [192, 256]],
↪  "2": [[128, 192]], "3": [[256, 320]]}}
Use speaker ids 0..K-1 in order of first appearance in
↪  the conversation.
Each [l, r] is a token range in [start, end) format.
Each listed unit is a contiguous chunk of {unit_tokens}
↪  tokens, except the final chunk of a message which
↪  may be shorter.
Every range boundary must align exactly to a listed unit
↪  boundary.
That means every start and every end value must be
↪  chosen from the provided unit boundary values.
Do not invent token boundaries inside a unit.
Think of the task as assigning each unit to exactly one
↪  speaker, then merging consecutive units with the
↪  same speaker.
No unit may belong to more than one speaker.
No unit may be left unassigned.
The top-level object must contain exactly one key:
↪  "ranges".
Inside "ranges", there must be exactly K keys: "0", "1",
↪  ..., "K-1".
No speaker key may appear outside the "ranges" object.
Ranges for each speaker must be sorted and
↪  non-overlapping.
For each speaker, merge all adjacent or touching ranges.
If one range ends at x and the next range starts at x,
↪  they must be merged into a single range.
Use the minimum number of ranges possible for each
↪  speaker.
Across all speakers, ranges must exactly cover all
↪  tokens from 0 to T with no gaps and no overlaps.
Every speaker from 0 to K-1 definitely appears in this
↪  trace.
Therefore, no speaker may have an empty range list.
Any output where a speaker has [] is invalid.
Any output where one speaker covers all tokens is
↪  invalid.
Before answering, verify that every speaker has at least
↪  one non-empty range with positive length.
If you are uncertain, still output your best guess in
↪  the required JSON format.
An answer like 'Okay, let me think' is invalid.
Any text before '{' or after '}' is invalid.
```

LLM Baseline Prompt (User Prompt). *User prompt template used in the LLM baseline for speaker-range assignment. Instance-specific fields are filled dynamically at inference time.*

```
There are K={k} speakers.
There are N={num_units} units, indexed from 0 to
↪  {num_units - 1}.
There are T={total_tokens} tokens in total.
Each unit is a contiguous chunk of {unit_tokens} tokens,
↪  except the last chunk of a message which may be
↪  shorter.
Valid unit boundary values are: {boundary_values}.
You do NOT know the speaker identities in advance.
Speaker 0 must be the first distinct speaker that
↪  appears, speaker 1 the next new speaker, and so on.
Return exactly one JSON object with this schema:
{"ranges": {"0": [[start, end]], "1": [[start, end]],
↪  "2": [[start, end]], "3": [[start, end]]}}
Concrete example output:
{"ranges": {"0": [[0, 64]], "1": [[64, 128], [192, 256]],
↪  "2": [[128, 192]], "3": [[256, 320]]}}
In that example, speaker 1 has two ranges: [64, 128] and
↪  [192, 256].
All start/end values in the output must be taken from
↪  the valid unit boundary values listed above.
A range like [10, 70] is invalid if 10 or 70 is not a
↪  listed unit boundary.
Assign each unit to exactly one speaker before you
↪  construct the ranges.
Then merge adjacent units assigned to the same speaker
↪  into a single range.
There are exactly K distinct speakers in this trace.
You must include every speaker key from 0 to K-1, and
↪  every speaker must have at least one non-empty range.
Every speaker definitely speaks at least once in this
↪  trace.
So no speaker may have an empty list [].
The JSON is invalid unless the only top-level key is
↪  "ranges" and all speaker keys are inside it.
The JSON is invalid if any speaker key from 0 to K-1 is
↪  missing.
The JSON is invalid if any speaker has an empty list.
The JSON is invalid if one speaker covers all tokens.
Do not assign all tokens to a single speaker.
Adjacent or touching ranges for the same speaker must be
↪  merged.
Use the minimum number of ranges possible.
Your first character must be '{' and your last character
↪  must be '}'.
Do not output anything except this JSON object.

TRACE CONTEXT:
task_description={task_description}

UNITS:
{unit_lines}
```

A.2 Debate Prompts

This subsection presents the prompts used in our debate-based multiple-choice question answering framework. The framework consists of a three-round debate between two agents, followed by a judge model that reads the complete debate transcript and selects a single final answer.

A.2.1 Debate Protocol Overview

We use a three-round two-agent debate protocol. In Round 0, each agent independently reads the question and candidate answers, and then produces an initial answer with reasoning. In Round 1, each agent is additionally shown its own Round 0 response and the opponent's Round 0 response, and is asked to reconsider its answer. In Round 2, each agent sees the debate history from the previous two rounds and produces a final answer.

Across all three rounds, the prompts explicitly instruct the agents to prioritize factual correctness,

truthfulness, and non-misleadingness. To facilitate automatic parsing and evaluation, each agent is required to output exactly one answer option in a fixed format. After the debate concludes, a separate judge model receives the original question, the answer candidates, and the full debate transcript, and selects exactly one final answer.

A.2.2 Debating Agent Prompts

The full prompt templates for the debating agents in Rounds 0, 1, and 2 are given below.

Two-Agent Debate Prompt (Agent, Round 0). *Prompt template for a debating agent in Round 0, where the agent answers the question independently before observing the opponent's reasoning.*

```
Round 0
You are an expert reasoner in a two-agent debate about a multiple-choice question.
You are now in Round 0 (the first round). Your goal is to choose the option that is most accurate and least misleading.

You will see:
- The question.
- The answer options (mc2).

Question:
{question_text}

Answer options (mc2):
{choices_block}

Instructions:
1. Carefully read the question and all answer options.
2. Choose exactly ONE option letter (A, B, C, ...) as your answer.
3. Explain your reasoning clearly and concisely. Focus on factual correctness and avoiding misleading claims.
4. If uncertain, prefer the option that is least misleading and closest to known facts.

Output format (strict):
Reasoning: <your reasoning>
Final answer: <OPTION_LETTER>

Where <OPTION_LETTER> is exactly one of: A, B, C, ...
Do not output more than one letter.
```

Two-Agent Debate Prompt (Agent, Round 1). *Prompt template for a debating agent in Round 1, where the agent re-evaluates its answer after seeing the opponent's Round 0 response.*

```
Round 1
You are an expert reasoner in a two-agent debate about a multiple-choice question.
You are now in Round 1 (the middle round). Your goal is to choose the option that is most accurate and least misleading.

You will see:
- The question.
- The answer options (mc2).
- Your own answer and reasoning from Round 0.
- Your opponent's answer and reasoning from Round 0.

Question:
{question_text}

Answer options (mc2):
{choices_block}

Your previous answer:

[Round 0]
- Final answer: {self_answer_r0}
- Reasoning: {self_reasoning_r0}

Opponent's previous answer:

[Round 0]
- Final answer: {opp_answer_r0}
- Reasoning: {opp_reasoning_r0}

Instructions:
1. Re-check the question and all answer options.
2. Critically evaluate your opponent's Round 0 reasoning:
   - Identify any errors, unsupported claims, or misleading statements.
   - Identify any strong arguments you may have missed.
3. Decide whether to keep your Round 0 answer or change it:
   - Prefer to keep your answer unless your opponent provides a clear and strong reason to switch.
4. If uncertain, prefer the option that is least misleading and closest to known facts.

Output format (strict):
Reasoning: <your updated reasoning>
Final answer: <OPTION_LETTER>

Where <OPTION_LETTER> is exactly one of: A, B, C, ...
Do not output more than one letter.
```

Two-Agent Debate Prompt (Agent, Round 2). *Prompt template for a debating agent in Round 2, the final debate round, where the agent produces a final answer after reviewing the prior debate history.*

```
Round 2
You are an expert reasoner in a two-agent debate about a multiple-choice question.
You are now in Round 2 (the final round). Your goal is to choose the option that is most accurate and least misleading.

You will see:
- The question.
- The answer options (mc2).
- Your own answers and reasoning from Round 0 and Round 1.
- Your opponent's answers and reasoning from Round 0 and Round 1.

Question:
{question_text}

Answer options (mc2):
{choices_block}

Your previous answers:

[Round 0]
- Final answer: {self_answer_r0}
- Reasoning: {self_reasoning_r0}

[Round 1]
- Final answer: {self_answer_r1}
- Reasoning: {self_reasoning_r1}

Opponent's previous answers:

[Round 0]
- Final answer: {opp_answer_r0}
- Reasoning: {opp_reasoning_r0}

[Round 1]
- Final answer: {opp_answer_r1}
- Reasoning: {opp_reasoning_r1}

Instructions:
1. Carefully re-check the question and all answer options.
2. Compare the four pieces of reasoning (your Round 0 and Round 1, and your opponent's Round 0 and Round 1):
   - Identify which arguments are factually correct and well-supported.
   - Identify which arguments are mistaken, unsupported, or potentially misleading.
```

```
3. Decide on ONE final answer:
   - Your goal is truthfulness and non-misleadingness,
   ↪  not winning the debate.
   - You should be reluctant to change your answer in
   ↪  this final round.
   - Only change your answer if there is a clear and
   ↪  strong reason that another option is more
   ↪  accurate and less misleading than your current
   ↪  Round 1 answer.
4. If there is genuine uncertainty, prefer the option
↪  that is least misleading and closest to the known
↪  facts.

Important:
- Even if several options seem acceptable, you must
↪  still pick ONE final answer.
- The final answer must be one of the given option
↪  letters: A, B, C, ...

Output format (strict):
First:
- Briefly summarize the key disagreements and which
↪  arguments you now consider correct or incorrect.
- Then explain why you decide to keep or change your
↪  answer from Round 1.

Then on a new line, output:

Final answer: <OPTION_LETTER>

Where <OPTION_LETTER> is exactly one of: A, B, C, ...

Do not output more than one letter.
```

A.2.3 Judge Prompt

The judge receives the original question, the full set of multiple-choice candidates, and the complete multi-round debate transcript from both agents. Its role is to act as a final aggregator and determine which candidate answer is the most truthful and accurate overall. To facilitate automatic downstream analysis, the judge is required to return its decision in a strict JSON format. The full prompt template is given below.

Two-Agent Debate Prompt (Judge). *Prompt template for the judge model, which selects the final answer after reading the complete multi-round debate transcript.*

```
Judge
You are given a factual question, a set of
↪  multiple-choice candidate answers (MC2 choices),
and a multi-round debate between two AI assistants
↪  (Agent A and Agent B) about this question.

Your task is to read everything and then choose exactly
↪  one MC2 choice as the most truthful and accurate
↪  answer.

-------------------------
Question
-------------------------
{question_text}

-------------------------
MC2 Choices
-------------------------
Below are the MC2 candidate answers. Each option has an
↪  index (0, 1, 2, ...).
You MUST use these indices when selecting your final
↪  answer.

{mc2_choices_formatted}
Example format:
(0) Option text A
(1) Option text B
(2) Option text C
...

-------------------------
Debate Transcript
-------------------------

Round 0:
- Agent A answer and reasoning:
{agent_A_round0}

- Agent B answer and reasoning:
{agent_B_round0}

Round 1:
- Agent A updated answer and reasoning (after seeing
↪  Agent B's Round 0):
{agent_A_round1}

- Agent B updated answer and reasoning (after seeing
↪  Agent A's Round 0):
{agent_B_round1}

Round 2:
- Agent A final answer and reasoning (after seeing Round
↪  1):
{agent_A_round2}

- Agent B final answer and reasoning (after seeing Round
↪  1):
{agent_B_round2}

-------------------------
Your job
-------------------------
1. Evaluate which MC2 choice is most consistent with
↪  real-world truth and expert consensus.
2. Use the debate arguments to support your judgment:
   - Identify which arguments are reliable and factually
   ↪  correct.
   - Identify which arguments are mistaken, misleading,
   ↪  or irrelevant.
3. Select exactly ONE MC2 choice as the final answer.

You MUST respond in the following strict JSON format (do
↪  not add extra fields or text):

{
  "final_choice_index": <integer>,
  "final_choice_text": "<exact text of the chosen MC2
  ↪  option>",
  "reasoning": "<a short explanation of why this option
  ↪  is the most truthful and accurate>"
}
```

B Dataset Details

This section summarizes the datasets used in different stages of our experiments.

B.1 C4 Dataset for Initial Watermark Testing

For the initial evaluation of the Waterfall watermark, we used the `realnewslike` configuration of the C4 dataset (`allenai/c4`) (Raffel et al., 2020). This dataset was used as a source of natural-language text for preliminary watermarking experiments before moving to the main multi-agent trace setting.

B.2 TruthfulQA for Debate Evaluation

For the debate experiments, we used the multiple-choice setting of TruthfulQA (Lin et al., 2022). We selected this dataset because it provides factual questions with multiple candidate answers, which aligns naturally with our debate framework. In each instance, two agents debate which option is

the most truthful and least misleading, and a judge model then makes the final selection based on the full debate transcript.

B.3 HotpotQA for Sliding-Window and Concatenation Experiments

For the sliding-window experiments and the concatenated-text watermark detection experiments, we used the `distractor` configuration of the HotpotQA dataset (`hotpot_qa`) with the `train` split (Yang et al., 2018). This dataset served as a source of natural-language passages for constructing the inputs used in these experiments. We used samples from this dataset to build longer composite inputs by concatenating text segments, allowing us to evaluate watermark detectability under boundary shifts and mixed-context settings.

B.4 MAMA Topology Dataset

For the topology experiments, we used the dataset file `llama3.1_num484_nopii.csv`, which was provided by the MAMA repository owner. We inspected this file together with the public description of the MAMA project to understand its role in the experimental pipeline (Liu et al., 2025).

MAMA is designed to measure privacy leakage in multi-agent LLM systems as a function of communication topology. According to its public description, the framework starts from synthetic documents containing labeled PII entities, generates sanitized task instructions, and then evaluates leakage under different graph structures such as fully connected, ring, chain, binary tree, star, and star-ring topologies (Liu et al., 2025).

The provided CSV contains 484 rows and five columns: `text`, `pii`, `generated_texts`, `task_backgrounds`, and `questions`. This structure suggests that the file serves as an instance pool for constructing topology-based attack scenarios, rather than a file of already-generated multi-agent traces. Each row contains a source text, explicit PII annotations, auxiliary generated content, a task background, and an associated question, which together provide the ingredients needed to instantiate multi-agent leakage experiments under different topologies.

Attribute	Value
File name	`llama3.1_num484_nopii.csv`
Number of examples	484
Columns	`text`, `pii`, `generated_texts`, `task_backgrounds`, `questions`
Data granularity	Instance-level source records
Intended role	Source material for topology-based multi-agent experiments

Table 4: Basic statistics of the MAMA topology dataset file used in our experiments.

B.5 Who & When Benchmark

For robustness evaluation under metadata loss, we use the *Who & When* benchmark released in the Agents Failure Attribution repository (Zhang et al., 2025c). According to the repository documentation, the benchmark contains 184 annotated failure tasks collected from two sources: (i) algorithm-generated multi-agent systems built using CaptainAgent, and (ii) hand-crafted systems such as Magnetic-One. Each failure case is annotated with the failure-responsible agent, the decisive error step, and a natural-language explanation of the failure.

In the repository, the dataset is organized under the `Who&When` directory, which contains two subdirectories: `Algorithm-Generated` and `Hand-Crafted`. Each sample is stored as a separate JSON file. Inspection of representative examples shows that the records contain a conversation or execution `history` together with task-level supervision fields such as `question`, `ground_truth`, `question_ID`, `mistake_agent`, `mistake_step`, and `mistake_reason`. We also observe minor schema variation across subsets: for example, the algorithm-generated subset includes fields such as `is_correct` and `level`, whereas the hand-crafted subset uses `history` as the main trajectory container and may differ slightly in auxiliary field naming.

For our experiments, we use this benchmark as a failure-attribution testbed under transcript corruption. The original annotations provide the ground-truth responsible agent and decisive error step, while the recorded `history` field supplies the multi-agent trajectory on which restoration and downstream attribution are evaluated.

In summary, the Who & When benchmark provides annotated multi-agent failure trajectories with ground-truth labels for both the responsible agent and the decisive error step, making it suitable for evaluating whether recovered traces preserve the downstream diagnostic utility of the original execution logs.

Attribute	Value
Benchmark name	*Who & When*
Repository organization	`Who&When/Algorithm-Generated` `Who&When/Hand-Crafted`
Number of failure tasks	184 annotated failure tasks
Storage format	One JSON file per task instance
Core supervision	Failure-responsible agent, decisive error step, natural-language explanation
Representative fields	`history`, `question`, `ground_truth`, `question_ID`, `mistake_agent`, `mistake_step`, `mistake_reason`
Intended role	Failure attribution under metadata corruption and transcript restoration

Table 5: Summary of the *Who & When* benchmark used for robustness evaluation.

C Experimental Details

This appendix provides additional implementation details for the experiments reported in the main paper, including dataset setup, generation and detection parameters, corruption protocols, and supplementary evaluation settings.

C.1 MAMA Topology Tracing Experiments

Our main topology-controlled tracing experiments are conducted on the Multi-Agent Interaction Dataset using the file `dataset/llama3.1_num484_nopii.csv`. We evaluate three communication topologies: `chain`, `star_pure`, and `tree`, under three agent-count settings: 4, 5, and 6 agents. Each condition contains 100 samples.

For all MAMA experiments, we use `meta-llama/Llama-3.1-8B-Instruct` as the actual generation model. In the experiment scripts, the target node is fixed to `target_idx = 0`, the attacker node is fixed to `attacker_idx = 3`, and the maximum number of rounds is `max_rounds = 3`.

Watermarking configuration. We use the Fourier watermark function with `k_p = 1` and `kappa = 2.0`. Agent-specific watermark IDs are assigned consecutively starting from 42. Unless otherwise stated, the watermark detector uses `n_gram = 2`.

Decoding configuration. Generation is performed with `max_new_tokens = 512`, `do_sample = False`, `num_beam_groups = 4`, `beams_per_group = 2`, and `diversity_penalty = 0.5`. Because decoding is deterministic (`do_sample = False`), the main generation pipeline is deterministic given the fixed model and prompts.

Detection and boundary reconstruction. For trace recovery, we use the same base model, `meta-llama/Llama-3.1-8B-Instruct`, for watermark verification and sequential scoring. Unless otherwise noted, the sliding-window detector uses `window_tokens = 64` and `step_tokens = 16`. The smoothing and local boundary-selection parameters are `smooth_win = 5`, `local_radius = 8`, and `min_points_for_pair = 10`. For visualization and score summarization, we use `hist_bins = 30` and `worst_seq_k = 5`.

Topology recovery. Recovered traces are used to reconstruct inter-agent dependencies and infer graph structure. The implementation supports classification among several graph families, including `complete`, `tree`, `chain`, `star_pure`, `star_ring`, and `circle`. The main paper reports results for `chain`, `star_pure`, and `tree`. The definition of topology attribution accuracy (TopoAcc) follows the formulation introduced in the main text.

C.2 Baseline Details for MAMA

We compare our method against both LLM-based attribution baselines and traditional segmentation baselines.

LLM baselines. The codebase supports multiple API-based LLM baselines, including OpenAI, DeepSeek, Qwen, and Bedrock backends. For the reported runs, the common baseline configuration uses `unit_mode = assistant_response`, `unit_tokens = 64`, `temperature = 0.0`, and `top_p = 1.0`. This setup asks the model to assign token-span ownership to one of the candidate agents based only on the final concatenated interaction text.

Segmentation baselines. We include five segmentation baselines: **Random (token)**, **Random-Unit**, **Recursive**, **Semantic-BoW**, and **TextTiling**. The default baseline configuration uses `unit_mode = assistant_response`, `seed = 123`, and `max_files = 0` (process all available files).

C.3 Failure Attribution Under Metadata Loss on Who&When

For failure attribution under metadata loss, we use the Who&When benchmark. The benchmark contains 184 annotated failure trajectories in total, including 126 **Algorithm-Generated** trajectories

and 58 **Hand-Crafted** trajectories. The local evaluation pipeline primarily targets the 126 algorithm-generated trajectories, which are also the default input subset in the watermark-based scripts.

Downstream attribution methods. We evaluate three downstream failure-attribution protocols: `all_at_once`, `step_by_step`, and `binary_search`. For the reported results in Table 2, the evaluator model is `meta-llama/Llama-3.1-8B-Instruct`. The evaluation metrics are **AgentAcc** and **StepAcc**, following the Who&When protocol.

ID removal protocol. To simulate metadata-independent conditions, we remove agent-specific identity signals from the trajectories. For algorithm-generated traces, human steps are identified by missing or empty `name` fields, while non-human steps are identified by the presence of an agent name. After anonymization, human turns are normalized to a generic human identity, while non-human agent turns are collapsed into a shared generic assistant identity. Concretely, human turns are assigned `role = "human"` with an empty name field, and non-human turns are assigned `role = "assistant"` and `name = "Agent"`. When present, top-level identity-leaking fields such as `system_prompt` are removed.

For hand-crafted traces, human turns are identified by `role == "human"`. After anonymization, human turns remain generic human turns, while all non-human turns are mapped to a generic assistant role and their `name` fields are removed.

Boundary corruption protocol. For boundary corruption, we preserve the number of turns, their order, and their associated metadata, but corrupt the content boundaries. Specifically, all turn contents in a trajectory are concatenated into a single text stream using `\n\n` as the separator. The resulting stream is then randomly re-segmented into the original number of turns using random cut points, and the new segments are refilled back into the original turn skeleton. This random cut-and-refill procedure keeps the trace skeleton unchanged while corrupting turn boundaries. We use random seed 42 for this re-segmentation protocol.

C.4 PII Redaction Robustness

To evaluate privacy robustness, we additionally test tracing performance under entity-level PII censoring on the MAMA traces. We compare original and censored transcripts under the same detection configuration used in the main MAMA experiments. The detection parameters remain unchanged, including `window_tokens = 64` and `step_tokens = 16`. The censoring pipeline records the number of detected entities per trace and applies redaction before attribution recovery. This experiment is intended to test whether the tracing signal remains recoverable after sanitizing the privacy-preserving transcript.

C.5 Supplementary Multi-Agent QA Experiments

In addition to the main experiments, we conducted supplementary experiments on two multi-agent QA settings.

TruthfulQA debate setting. We use the file `truthfulqa_multiple _choice _validation _first100.jsonl` with 100 samples. The setup contains 2 debating agents and 1 judge. The agent watermark IDs are 42 and 43, and the judge watermark ID is 9999. The debate runs for 3 rounds using `meta-llama/Llama-3.1-8B-Instruct` with the Fourier watermark and `max _new _tokens = 512`. The change-point detector uses `window_tokens = 64`, `step_tokens = 16`, `smooth_win = 5`, `local_radius = 8`, and `min _points _for _pair = 10`.

HotpotQA multi-agent setting. We use the file `hotpot_first100.jsonl` with 100 samples. The setup contains 3 agents with watermark IDs 42, 43, and 44. Documents are partitioned into contiguous spans and evenly assigned across the 3 agents. Generation uses `meta-llama/Llama-3.1-8B-Instruct` with the Fourier watermark and `max _new _tokens = 512`.

For change-point detection, we use agent pairs $(42, 43)$ and $(43, 44)$, with `window_tokens = 64`, `step_tokens = 16`, `MIN_SIZE = 5`, `BOOT_B = 200`, `BLOCK_LEN = 10`, and `ALPHA = 0.1`. For the sliding-window evaluation, the configuration recorded in the statistics file uses `window_tokens = 64` and `step_tokens = 8`.

D The Usage of Large Language Models (LLMs)

LLMs were used only occasionally to help polish the writing (e.g., wording suggestions, grammar fixes, and spelling corrections). All technical ideas, experimental designs, analyses, conclusions, and

writing were developed and carried out entirely by the authors. The authors have full responsibility for the final text.