VerifyMAS: Hypothesis Verification for Failure Attribution in LLM-based Multi-agent Systems

Hezhe Qiao[1], Hanghang Tong[2], Ee-Peng Lim[1], Bing Liu[3], Guansong Pang[1]*
[1]Singapore Management University
[2]University of Illinois at Urbana-Champaign
[3]University of Illinois at Chicago
hezheqiao.2022@phdcs.smu.edu.sg
htong@illinois.edu, liub@uic.edu
{eplim,gspang}@smu.edu.sg

Abstract

Large language model-driven multi-agent systems (LLM-MAS) excel at complex tasks, yet unreliable agents remain a key bottleneck to system-level reliability. Automatic failure attribution is therefore critical, but existing approaches—such as direct prediction of agent–error pairs and agent-first failure attribution—rely on local logs of agent and miss global failures that only manifest over full interaction trajectories, such as cross-step inconsistencies and inter-agent coordination errors. Moreover, directly predicting failures induces a large combinatorial search space, hindering fine-grained attribution. To address these challenges, we propose **VerifyMAS**, a hypothesis verification framework for agent failure attribution. Instead of directly predicting faulty agents and error types, VerifyMAS formulates and verifies failure hypotheses against full trajectories. This verification-based approach decomposes attribution into trajectory-level error validation and fine-grained agent localization, providing **an error-first attribution approach** that captures global failure patterns while substantially reducing the search space. We further introduce a hypothesis-based data construction strategy grounded in a structured error taxonomy and fine-tune a specialized LLM verifier model for trajectory-level failure verification and agent attribution. Experiments on *Aegis-Bench* and *Who&When* show that VerifyMAS consistently improves diverse backbone models, including open-source Qwen and API-based GPT and Gemini models, outperforming prior methods without sacrificing inference efficiency for long multi-agent trajectories. Code and data are available at https://github.com/mala-lab/VerifyMAS.

1 Introduction

Large language model-driven multi-agent systems (LLM-MAS) have recently attracted significant attention due to their strong potential for enabling multiple agents to collaboratively solve complex tasks, such as reasoning, planning, decision-making, and tool use [24, 7, 18, 16]. By decomposing a task into several subtasks and assigning them to specialized agents, LLM-MAS can often achieve better flexibility and efficiency than single-agent systems [30, 37]. However, this collaborative setting also introduces new challenges. Because the final outcome depends on interactions among multiple agents, failures in a single agent can propagate through the whole system, and eventually leading to incorrect or suboptimal outcomes.

*Corresponding author: G. Pang

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

In practice, agent failures can arise in various forms [31, 4, 35]. For example, an agent may deviate from task requirements, misinterpret its assigned role, overlook critical information from other agents, produce inconsistent reasoning, or terminate the process prematurely. These failures are often difficult to identify because they are often embedded within long interaction trajectories and may not be immediately observable in intermediate outputs [14, 3]. Consequently, improving the reliability of LLM-MAS requires not only stronger collaboration strategies but also effective mechanisms for failure attribution [4, 35].

Failure attribution has emerged as a central task in guaranteeing the usefulness of LLM-MAS, focusing on identifying which agent caused a failure and classifying the types of errors that led to it. Existing methods typically formulate it as a direct error-agent pair prediction problem, in which the predictions of error types and the faulty agent are combined into a single prediction task [13, 32], as shown in Fig. 1**a**. While simplifying the problem, this formulation often i) overlooks explicit evidence grounding, ii) struggles to capture long-range dependencies across interaction steps, and iii) tends to produce surface-level error labels without sufficiently verifying whether the failure truly occurred or materially affected the final outcome. Moreover, this approach induces a large combinatorial search space of agents and failure categories, hindering fine-grained attribution. Some recent work [13, 16, 32] tackles this challenge using an agent-first attribution approach, which leverages chain-of-thought (CoT) prompting to first identify the responsible agent and then attribute the corresponding error, but this approach can easily bias the model toward local interaction context since each agent's logs provide only partial observations of the full trajectory.

To address these challenges, we propose **VerifyMAS**, a hypothesis verification framework for agent failure attribution. The key idea is to augment LLMs with a set of hypotheses over predefined failure modes to automatically verify what type of error has occurred in a multi-agent trajectory and subsequently determine which agent is responsible. To this end, given a full trajectory, VerifyMAS verifies whether the trajectory entails, is neutral to, or contradicts each of the hypotheses describing the presence of an error type, yielding one of three possible outcomes: "*entail*", "*neutral*", and "*contradict*", as shown in Fig. 1**a**. For entailed hypotheses, VerifyMAS further performs agent attribution to identify the responsible agent(s). This decomposes the failure attribution into trajectory-level error validation and fine-grained faulty agent localization, thereby avoiding the combinatorial complexity from reasoning over the faulty agents and error types in a joint manner.

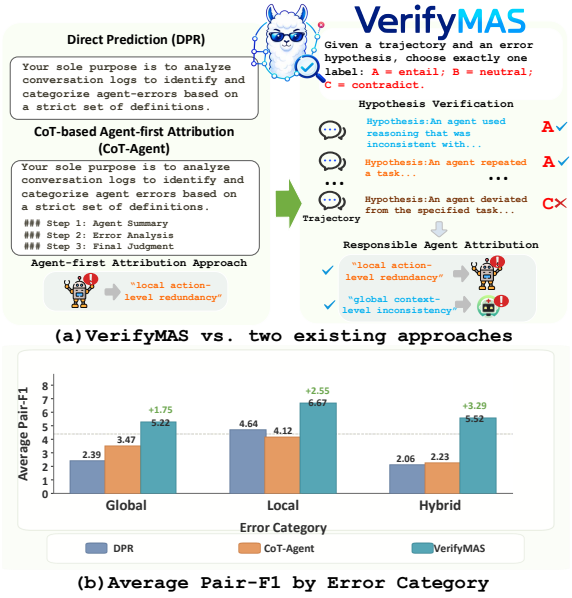

Figure 1: (a) Our hypothesis verification-based approach VerifyMAS vs. two existing approaches. (b) Performance of three approaches on Aegis-Bench [13] in attributing three categories of error types—global, local, and hybrid errors in average Pair-F1.

VerifyMAS yields an error-first attribution methodology that prioritizes the verification of error types against the full trajectory, enabling accurate attribution of **global errors** that only manifest over the interaction log sequences of two or more agents, as shown in Fig. 1**b**. This contrasts fundamentally with existing agent-first methods that assess failures at the individual-agent level and are therefore prone to biases induced by local agent behaviors. Consequently, they often work well to attribute **local errors** that are observable at individual agents, but become ineffective to attribute global errors, having even worse performance in **hybrid errors** that require both agent-level local behavioral evidence and trajectory-level global context, as shown in Fig. 1**b** (see App. B for detailed examples of these error categories). More importantly, our error-first approach VerifyMAS can also perform very well on attributing local and hybrid errors. In summary, this work makes the following three main contributions.

- We propose **VerifyMAS**, an error-first hypothesis verification framework for failure attribution in LLM-MAS. It decomposes failure attribution into error validation and fine-grained faulty agent localization, providing a principled solution for agentic failure attribution of both global and local errors.
- We propose a fine-tuning strategy tailored to the hypothesis verification approach, in which trajectory-level verification samples and agent-localization supervision are collected and leveraged to fine-tune an LLM verifier model under the VerifyMAS framework. This substantially enhances our model in failure diagnosis of in-distribution trajectories while preserving robust generalization to out-of-distribution trajectories. The dataset will be released to promote more advances in this line.
- Extensive experiments on *Aegis-Bench* and *Who&When* demonstrate that VerifyMAS consistently improves diverse open-source and proprietary models. We further validate its effectiveness under the SFT setting, where hypothesis-verification-based fine-tuning strengthens in-distribution diagnostic ability while preserving out-of-distribution generalization.

2 Related Work

Safeguarding MAS LLM-based guard models have recently been studied as lightweight classifiers for monitoring and filtering model misbehaviors [25, 9, 15, 8]. Recent work extends this idea to agentic systems, where evaluators need to inspect intermediate steps rather than only final answers [36, 26, 12, 33]. Early representative systems, such as Llama Guard [11], ShieldGemma [28], and Aegis [8] mainly focus on content safety moderation, where a guard model classifies user inputs or model responses according to a predefined risk taxonomy. These methods demonstrate that instruction-tuned LLMs can serve as flexible taxonomy-guided classifiers. Different from these guard models that perform content-level safety classification of isolated prompts or responses, our setting focuses on the analysis of complete multi-agent interaction trajectories to attribute failures.

Another line of work studies guard models from the perspective of anomaly detection [19, 34, 17, 2], aiming to detect behaviors (*e.g.*, failures) that deviate from normal or expected MAS patterns. The task is orthogonal to failure attribution that provides detailed root cause analysis of the detected failures. Additionally, they focus on learning representation-level deviation patterns, while our setting require contextual semantic understanding of the tasks assigned to MAS and their full trajectory.

MAS Failure Attribution Failure attribution has recently emerged as a critical task for safeguarding the reliability of LLM-MAS. Early studies [4, 13, 32, 35, 16] introduce predefined taxonomies of failure modes and formulate failure attribution as direct agent-error pair prediction, jointly determining both the error type and the responsible agent within a single prediction task. However, directly predicting agent-error pairs introduces a large combinatorial search space are struggle to capture long-range dependencies. Some recent works [13, 32, 16, 29] further leverage chain-of-thought prompting to improve failure attribution by first analyzing the behaviors of each agent and then identifying possible failures, forming an agent-first attribution approach. Although such methods can provide more explicit intermediate analysis compared with direct prediction, starting from individual agents may bias the model toward local action-level mistakes, while overlooking trajectory-level evidence that emerges only from cross-step interactions or coordination failures [23, 12, 4, 13].

Different from prior work that directly predicts the responsible agent-error pair and agent-first attribution approach, we formulate agent failure attribution as a grounded hypothesis-verification problem over long multi-agent trajectories, enabling finer-grained reasoning over specific error-agent types and explicit support/contradiction signals from the whole trajectory evidence with global context instead of local agent-level behavioral evidence.

3 Our Approach: VerifyMAS

Problem Statement An LLM-MAS interaction trajectory is defined as $\tau = (I, l_1, \ldots, l_T)$, where I represents the initial task instruction, $l_1, l_2, \cdots, l_T$ denote a collection of log sequences produced by a finite set of K LLM agents $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ over T interaction steps to solve a given task, with each l_i denoting a log sequence that traces the actions and reasoning process of a specialized agent a_j taken at step i. An agent can be assigned to work in different steps, so we typically have $T \gg K$.

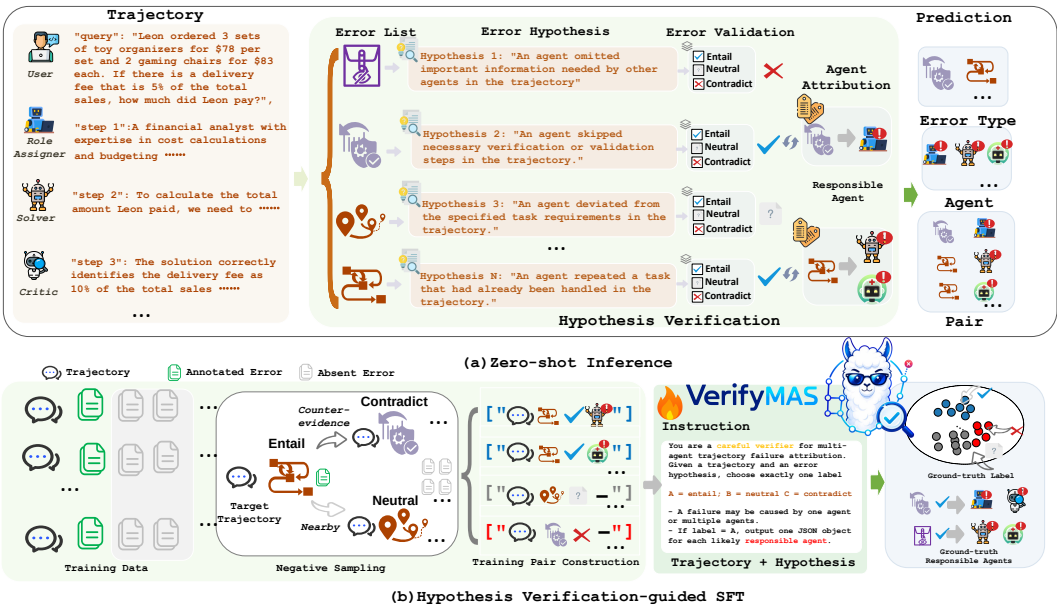

Figure 2: Overview of the proposed VerifyMAS. **(a)** Zero-shot inference. A trajectory is paired with hypotheses on predefined error types, and then an LLM predicts whether the trajectory is "entail", "neutral", or "contradict" w.r.t. each hypothesis describing the presence of an error type. The entailed hypotheses are further examined for faulty agent attribution, producing the final error–agent predictions. **(b)** Hypothesis verification-guided supervised fine-tuning. We construct trajectory–hypothesis pairs by pairing each trajectory with both annotated errors and absent errors. VerifyMAS is then fine-tuned to jointly predict the type of the error and the responsible agent.

Given a trajectory of an LLM-MAS failure τ, our objective is to attribute the failure by determining which agent caused a specific error that led to the failure of the overall agent system. The error space is defined by a predefined taxonomy of M distinct error types, denoted as $\mathcal{Y} = \{y_1, \ldots, y_M\}$. For a failure trajectory, the ground-truth annotation is represented as a structured set of agent–error pairs, denoted by $\mathcal{G}(\tau) = \{(a_1^*, y_1^*), (a_2^*, y_2^*), \ldots\}$, where $y_i^* \in \mathcal{Y}$ denotes the error mode and a_j^* denotes the agent responsible for causing the error. A single trajectory can contain either single or multiple agent–error pairs as its ground truth, *i.e.*, one trajectory may have multiple labels. The core task is to obtain a diagnostic function f_θ that maps a trajectory τ to a predicted attribution set that approximates the ground truth: $f_\theta : \tau \mapsto \hat{\mathcal{G}}(\tau)$.

Overview of VerifyMAS As shown in Fig. 2, our proposed VerifyMAS framework can operate under either zero-shot inference or supervised fine-tuning (SFT). In the zero-shot inference setting, given a multi-agent trajectory, we first instantiate a set of hypotheses based on the predefined failure taxonomy. Each hypothesis is paired with the trajectory and evaluated by an LLM verifier, which predicts one of three labels: entail, neutral, or contradict. Hypotheses that are labeled as entail are retained as candidate failures, upon which the model further performs faulty agent attribution, identifying the most likely responsible agent for each supported error hypothesis. In SFT, we leverage the trajectories with annotated faulty agents and error types to construct training instances by pairing each trajectory with error hypotheses and assigning labels. For an entailed hypothesis, the corresponding responsible agents are used as supervision for agent attribution, while neutral and contradictory hypotheses are assigned an empty agent label. These trajectory–hypothesis pairs, together with their error type and agent annotations, are then used to perform SFT of our LLM verifier, so that the verifier learns to jointly verify hypotheses and attribute responsibility to the faulty agent.

3.1 Zero-shot Failure Attribution

Error Hypothesis Formulation Instead of directly predicting multiple error labels from the entire trajectory, we reformulate agent failure attribution as a grounded hypothesis verification problem. This enforce the model to skip the local context of the agent and evaluate the trajectory from a global

perspective. We construct several natural-language hypotheses that only ask whether a specific error type $y_m \in \mathcal{Y}$ is supported by the trajectory, without assigning it to any particular agent. Formally, for each error type y_m, we construct a hypothesis h_m describing that some agent in the trajectory exhibits error type y_m. For example, for the error type *task specification deviation*, the corresponding hypothesis is instantiated as: *An agent deviated from the specified task requirements in the trajectory*.

Error-hypothesis Verification Based on the above formulation, we perform failure attribution in two stages. In the first stage, we verify all type-level hypotheses $\{h_m\}_{y_m \in \mathcal{Y}}$ against the trajectory to identify candidate error types. This stage is designed to identify failure modes that can only be reliably recognized from global trajectory-level evidence rather than directly traversing agents and judging errors from local agent logs. This is more suitable for detecting errors that emerge from cross-step dependencies, long-range context, or multi-agent interactions. Formally, we formulate this step as a three-way hypothesis validation problem, where the predicted label is selected from

$$z_m \in \{A, B, C\},$$

with A, B, and C denoting *entail*, *neutral*, and *contradict*, respectively. Specifically, A indicates that the trajectory provides clear evidence for the hypothesized failure, B indicates that the evidence is insufficient or ambiguous, and C indicates that the hypothesis is contradicted by the trajectory. We adopt a three-way formulation rather than a binary one because many failure hypotheses cannot be confidently verified or rejected from the trajectory alone. The *neutral* class captures cases with weak, incomplete, or ambiguous evidence, as well as errors whose impact on the final outcome is unclear. This avoids forcing uncertain cases into either positive or negative labels and leads to more reliable failure verification. Formally, given a trajectory τ_i and hypothesis h_m for error type y_m, the verifier predicts the label for error $\hat{z}_{i,m}$ by

$$\hat{z}_{i,m} = \arg \max_{z \in \{A,B,C\}} f_\theta(z \mid \tau_i, h_m, P, \mathcal{A}_i),$$

where P denotes the instruction prompt that guides the LLM to verify whether the given hypothesis is supported by the trajectory τ_i, and $\mathcal{A}_i$ is the candidate agent set for τ_i. Only hypotheses predicted as entail are passed to the agent attribution stage. Therefore, the retained failures for τ_i are

$$\widehat{\mathcal{Y}}_i = \{y_m \mid \hat{z}_{i,m} = A\},$$

where $\widehat{\mathcal{Y}}_i$ are selected error modes from all the error-hypothesis verification for trajectory τ.

Agent-level Attribution After hypothesis validation, agent attribution is performed only for hypotheses with label A. Note that different trajectories may contain different numbers and names of agents. For each retained error type $y_m \in \widehat{\mathcal{Y}}$, the model identifies which candidate agents are responsible for the entail failure. The responsible agents must be selected from the current candidate set $\mathcal{A}_i$, rather than from a fixed global agent vocabulary.

For cases where multiple agents are responsible for the same entail failure, the model outputs one JSON object for each responsible agent, with each object containing exactly one agent. The final attributed failure set for trajectory τ_i is then written as

$$\widehat{\mathcal{G}}(\tau_i) = \{(a_n, y_m) \mid y_m \in \mathcal{Y},\ a_n \in \mathcal{A}_i\},$$

where $\widehat{\mathcal{G}}(\tau_i)$ denotes the predicted responsible agent-error pair where each tuple (a_n, y_m) indicates that agent a_n is inferred to be responsible for the entail error type y_m.

This two-stage design separates failure validation from agent localization. Our key insight is that many failure modes are not evident from a single agent's local log, but only emerge when considering the full trajectory, including cross-step dependencies, inter-agent interactions, and the final task outcome. Therefore, we first start from the error side and verify whether each hypothesized failure is supported by the global trajectory context. Meanwhile, conditioning agent attribution on validated failures allows the model to focus on agents whose actions are relevant to the detected error, making it easier to identify problematic inter-agent interactions.

3.2 Hypothesis Verification-guided Supervised Fine-tuning

Training Data Construction We construct the supervised fine-tuning dataset from annotated multi-agent trajectories. Each trajectory is associated with ground-truth faulty agents and their corresponding error types. For each trajectory, we first generate entailment examples using the ground-truth

annotations. Specifically, if an agent a is annotated as responsible for an error type y_m, given the input $\mathbf{x}_{i,m} = \{\tau_i, h_m, P, \mathcal{A}_i\}$ with trajectory corresponding to error hypothesis h_m , instruction prompt P, and the candidate agent list $\mathcal{A}_i$ in the trajectory, we construct a positive example with label $entail$ as the corresponding output:

$$\mathbf{y}_{i,m} = \{\texttt{"label"} : \texttt{"A"}, \texttt{"agent"} : [a]\}, \hat{z}_{i,m} = A,$$

where $a \in \mathcal{A}_i$ cause the error y in the trajectory. If multiple agents are responsible for the same entailed error hypothesis, we construct multiple JSON objects, each corresponding to one responsible agent and labeled as entail. These positive examples teach the model to recognize when a hypothesized failure is supported by the trajectory and to attribute it to the responsible agent.

To construct contradict and neutral examples, we sample errors from absent error *i.e.*$\mathcal{Y}_a = \mathcal{Y}/\mathcal{Y}_e$ where $\mathcal{Y}_e$ is the set of existing annotated errors, to formulate the negative samples for this trajectory. For contradicted examples, we sample errors from absent error types that are explicitly refuted by trajectory evidence and semantically incompatible with the ground-truth failure annotation to formulate the hypotheses. Specifically, we construct an explicit counter-evidence table for each error type, see App. D.1, which specifies evidence cues whose presence in the trajectory indicates that the corresponding error is unlikely to have occurred. Therefore, when such cues are observed, the associated absent-error hypothesis can be treated as a contradicted hypothesis.

For neutral examples, we sample absent error types following a nearby error-type mapping table, see App. D.1, which groups semantically related or easily confusable error types according to their definitions and behavioral features. As a result, neutral examples often correspond to partially plausible agent behaviors or related failure modes for which the trajectory provides insufficient evidence to make a confident entailment decision. These two types of negative samples provide fine-grained negative supervision that helps the verifier distinguish clearly contradicted hypotheses from plausible but unsupported neutral hypotheses. The corresponding outputs for the contradict $\mathbf{x}_{contradict}$ and neutral $\mathbf{x}_{neutral}$ input with hypothesis h_m are

$$\begin{cases} \mathbf{y}_{\text{neutral}} = \{\texttt{"label"} : \texttt{"B"}, \texttt{"agents"} : []\}, & \hat{z} = B, \\ \mathbf{y}_{\text{contradict}} = \{\texttt{"label"} : \texttt{"C"}, \texttt{"agents"} : []\}, & \hat{z} = C. \end{cases}$$

Training Loss Function We formulate the hypothesis-verification approach based SFT as a unified structured-output generation problem. For each instance $\mathbf{x}_{i,m}$, which contains the trajectory τ_i, the error hypothesis h_m, the candidate agent set $\mathcal{A}_i$, and the instruction P, the model is trained to generate a serialized JSONL-style response. The target response $\mathbf{y}_{i,m}$ contains both the verification label and the responsible agents.

When multiple agents are responsible, the target is serialized as multiple JSON objects, one for each responsible agent; when the hypothesis is neutral or contradict, the target is a single JSON object with an empty agent list. This design naturally supports instance-specific candidate agent sets and avoids introducing a fixed global agent classifier. Specifically, given the input $\mathbf{x}_{i,m}$, after serializing the target JSON object(s) into a response string and tokenizing it, we denote the gold response as $\mathbf{y}^*_{i,m} = (y^{(1)*}_{i,m}, y^{(2)*}_{i,m}, \ldots, y^{(|\mathbf{y}^*_{i,m}|)*}_{i,m})$, where $y^*_{i,t}$ denotes the t-th token in the serialized target response and $|\mathbf{y}^*_{i,m}|$ is the number of response tokens. The training loss $\mathcal{L}_{\text{HSFT}}$ for $\mathbf{x}_{i,m}$ is

$$\mathcal{L}_{\text{HSFT}} = -\sum_{t=1}^{|\mathbf{y}^*_{i,m}|} \log p_\theta \left(y^{(t)*}_{i,m} \mid \mathbf{x}_{i,m}, y^{(<t)*}_{i,m} \right),$$

In this way, the model is supervised to generate structured decisions that first verify whether the hypothesis is entailed and then output the corresponding responsible agent(s) when applicable.

4 Main Experiments

Datasets. Following previous work [13, 32], we evaluate all methods on two benchmark datasets: Aegis-Bench [13] and Who&When [32]. Aegis-Bench [13] is a trajectory dataset that was collected from six task domains and six multi-agent system frameworks. Who&When [32] is a failure log dataset collected from 127 LLM multi-agent systems with fine-grained annotations linking failures to

Table 1: Performance gains of VerifyMAS-enabled models over their corresponding base models. The best performance is boldfaced. Arrows indicate changes over the base model, where ↑ denotes improvement and ↓ denotes degradation.

Model	Aegis-Bench						Who&When					
	Pair		Agent		Error		Pair		Agent		Error	
	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1
Open-source Models												
AgentDebug*	4.34	0.49	45.90	**16.29**	11.53	2.25	3.47	1.81	37.45	33.62	9.25	1.54
AgentDoG*	0.83	0.11	13.01	4.07	3.34	1.55	0.90	0.25	12.67	9.13	3.70	2.45
Qwen2.5-7B-Instruct	5.02	2.52	27.55	14.49	14.96	11.36	2.31	1.14	40.92	23.50	3.64	1.77
+VerifyMAS	5.89 ↑0.87	**2.98** ↑0.46	47.92 ↑20.37	14.66 ↑0.17	25.07 ↑10.11	24.26 ↑12.90	5.83 ↑3.52	**3.44** ↑2.30	39.26 ↓1.66	33.50 ↑10.00	13.33 ↑9.69	9.54 ↑7.77
Qwen3-8B-Non-Thinking	3.96	1.40	21.34	8.16	15.81	13.89	3.88	1.81	27.78	17.64	3.88	1.91
+VerifyMAS	5.49 ↑1.53	1.85 ↑0.45	42.96 ↑21.62	11.65 ↑3.49	26.50 ↑10.69	24.05 ↑10.16	4.92 ↑1.04	2.88 ↑1.07	35.03 ↑7.25	24.11 ↑6.47	11.93 ↑8.05	5.48 ↑3.57
Qwen2.5-14B-Instruct	5.47	2.20	35.78	12.71	20.24	5.91	0.00	0.00	**49.88**	33.19	1.56	1.35
+VerifyMAS	**7.40** ↑1.93	2.69 ↑0.49	**48.95** ↑13.17	14.50 ↑1.79	**28.37** ↑8.13	**26.90** ↑20.99	5.15 ↑5.15	2.68 ↑2.68	35.97 ↓13.91	27.53 ↓5.66	10.53 ↑8.97	7.04 ↑5.69
Qwen3.5-9B-Non-Thinking	3.35	1.02	25.21	9.26	10.73	8.63	5.18	2.86	45.28	35.60	9.73	4.33
+VerifyMAS	5.51 ↑2.16	2.18 ↑1.16	41.65 ↑16.44	15.81 ↑6.55	27.15 ↑16.42	26.30 ↑17.67	**6.16** ↑0.98	3.01 ↑0.15	46.63 ↑1.35	**36.42** ↑0.82	**15.45** ↑5.72	**10.74** ↑6.41
Proprietary Models												
AgentDebug**	5.41	0.58	44.01	14.71	12.64	2.68	4.57	2.68	44.70	38.04	9.30	4.31
AgentDoG**	6.19	0.93	52.01	17.94	14.69	5.80	5.00	2.62	43.05	37.52	**15.15**	6.55
GPT-4o-mini	5.76	1.63	38.54	14.72	19.95	16.02	2.11	0.98	47.42	34.21	5.26	3.33
+VerifyMAS	6.58 ↑0.82	2.62 ↑0.99	49.30 ↑10.76	16.54 ↑1.82	27.89 ↑7.94	**27.79** ↑11.77	5.19 ↑3.08	3.19 ↑2.21	44.96 ↓2.46	38.00 ↑3.79	12.39 ↑7.13	11.36 ↑8.03
GPT-4.1	7.44	2.27	37.48	11.12	20.65	15.75	3.36	1.16	42.29	28.93	7.00	5.84
+VerifyMAS	**7.59** ↑0.15	2.79 ↑0.52	49.52 ↑12.04	17.07 ↑5.95	**27.95** ↑7.30	27.37 ↑11.62	5.09 ↑1.73	2.83 ↑1.67	42.93 ↑0.64	37.44 ↑8.51	13.06 ↑6.06	10.66 ↑4.82
Gemini-2.5-Flash	6.99	2.76	42.02	16.45	23.47	19.85	7.32	3.33	**55.56**	36.98	11.94	7.96
+VerifyMAS	7.02 ↑0.03	2.77 ↑0.01	50.33 ↑8.31	16.91 ↑0.46	27.23 ↑3.76	26.73 ↑6.88	**7.39** ↑0.07	3.49 ↑0.16	43.25 ↓12.31	37.23 ↑0.25	13.33 ↑1.39	**11.65** ↑3.69
Gemini-2.5-Pro	6.96	**2.88**	41.32	16.15	19.93	16.29	6.81	2.69	53.11	34.92	11.07	8.11
+VerifyMAS	7.47 ↑0.51	2.60 ↓0.28	**53.61** ↑12.29	**18.21** ↑2.06	27.27 ↑7.34	26.99 ↑10.70	7.33 ↑0.52	**4.18** ↑1.49	49.10 ↓4.01	**41.47** ↑6.55	12.94 ↑1.87	11.39 ↑3.28

specific agents and decisive error steps. It is used as an out-of-distribution (OOD) benchmark for evaluation. Details information about datasets can be found in App. B.

Evaluation Metrics. Following prior work [13, 32], we employ Micro-F1 (μF1) and Class-wise Macro-F1 (MF1) as the primary metrics. In particular,μF1 emphasizes overall classification performance, whereas MF1 highlights performance across different classes by giving equal weight to each class. Because a single trajectory may involve multiple faulty agents and multiple error modes, we evaluate performance at three levels of granularity: **Pair**-level, which measures correctly identified agent–error pairs; **Agent**-level, which measures faulty-agent identification regardless of error type; and **Error**-level, which measures error-mode identification regardless of the responsible agent. In addition to F1, recall and precision are also provided as auxiliary metrics.

Competing Methods and Implementation Details. We compare VerifyMAS with not-formally-published approaches addressing closely related problems, AgentDebug [35] and AgentDoG [16] and a range of open-source models, including Qwen models at different scales: 7B-Instruct, 8B, 9B in non-thinking, and 14B-Instruct [27, 21, 22]. We also compare with Proprietary models, including GPT-4.1 [1], GPT-4o-mini [10], Gemini-2.5-Flash [5], and Gemini-2.5-Pro [5]. All these models are evaluated under a zero-shot setting with the instruction prompt provided in Aegis [13]. We implement VerifyMAS over the base model, with the detailed instruction prompt provided in the App. F. The same task instructions are used in our SFT models. In addition, we compare our SFT method against the contrastive-based DCL model [13] and several open-source baselines under SFT setting [27, 21].

Training Details. For our SFT experiments, we utilize the verl library [20], and all fine-tuning is conducted on a cluster of 2 NVIDIA H200 GPUs. The learning rate is set to $1e-4$, and the batch size is set to 64. VerifyMAS is trained for 2 epochs based on Qwen2.5-7B-Instruct and Qwen3-8B. More details of hyperparameters for all SFT runs can be found in the App. D.2.

4.1 Zero-shot Results

To examine the effect of VerifyMAS, we first implement VerifyMAS using several open-source models and proprietary models, then we conduct a zero-shot evaluation on the AEGIS and Who&When benchmarks, comparing it against their base models. The main results are shown in Table 1. The detailed recall and precision results can be found in the App. C. From the results, (1) we observe that applying VerifyMAS yields a substantial absolute gain compare the original open-source model. On Aegis-Bench, VerifyMAS can consistently improve performance across Pair, Agent, and Error levels, with particularly large gains on Agent μF1 and Error MF1. This indicates that starting from the error hypothesis verification enables the model to better capture global trajectory-level errors and provides a more reliable basis for subsequent agent localization. On the Who&When benchmark,

VerifyMAS leads to a slight drop in agent-level identification on 7B-Instruct and 14B-Instruct, while an improvement was observed on the 8B model, mainly because different models vary in their ability to attribute verified failures to the correct agents. Overall, VerifyMAS consistently improves the overall average score at the pair and failure levels across multiple Qwen models, indicating stronger generalization to unseen trajectory distributions. (2) VerifyMAS can also benefit stronger proprietary models. These results show that applying VerifyMAS to proprietary models like GPT-4.1 and Gemini-2.5-Pro can improve the average score across three levels. This further demonstrates that VerifyMAS is model-agnostic and can consistently enhance both open-source and closed-source models. The gains are especially pronounced at the Error level, suggesting that hypothesis-guided trajectory-level verification is effective for detecting failure modes that require global context, cross-step dependencies, and inter-agent interaction evidence.

Comparison with AgentDoG and AgentDebug. To further demonstrate the effectiveness of VerifyMAS, we compare VerifyMAS with two concurrent, not-formally-published approaches addressing closely related problems, AgentDebug [35] and AgentDoG [16]. We adapted both methods to our evaluation setting and included them as additional SOTA baselines. For a comprehensive and fair comparison, we implemented both baselines with Qwen2.5-7B (AgentDebug* and AgentDoG*) and GPT-4o-mini (AgentDebug** and AgentDoG**). We then compared each baseline against the corresponding VerifyMAS variant using the same backbone model. The results show that VerifyMAS (with either Qwen2.5-7B or GPT-4o-mini) significantly outperforms both AgentDebug and AgentDoG at both the error-type and agent–error pair levels on the two datasets, e.g., achieving up to 5.0+ (Pair μF1) and 20.0+ (Error μF1/MF1) improvement. This advantage stems from the fact that these two baselines are primarily designed for step-level or agent-level failure detection rather than fine-grained failure attribution. Although the error-first principle renders VerifyMAS less advantageous in agent-level performance, benefiting from its progressive and stringent verification, VerifyMAS remains highly competitive to these two methods in both agent-level metrics.

4.2 Supervised Fine-tuning Results

To further demonstrate the effectiveness of our approach, we conduct experiments under the SFT setting on the Aegis-Bench and Who&When benchmarks. In this setting, VerifyMAS is trained on the constructed error hypothesis verification training data and compared with local Qwen-based models.

As shown in Table 2, VerifyMAS-7B consistently achieves the best average performance on both benchmarks, demonstrating its effectiveness and better generalization ability compared with both the Qwen2.5-7B-SFT and the supervised DCL baseline. The improvement of VerifyMAS mainly comes from better pair-level attribution and error-type recognition. On Aegis-Bench, VerifyMAS-7B-SFT and 8B-SFT achieve the best Pair-F1 scores, indicating that hypothesis-guided verification helps the model identify both the failure category and the corresponding agent-error pair more accurately. Similar trends can be observed on Who&When, where VerifyMAS-7B-SFT obtains the strongest Pair-level μF1 performance and VerifyMAS-8B-SFT yields the highest Pair-level MF1. Note that its Agent-level performance is not always the best on a specific metric, suggesting that accurately localizing the responsible agent remains challenging, especially when multiple agents interact over long trajectories.

Table 2: SFT performance comparison.

Model	Pair		Agent		Error	
	μF1	MF1	μF1	MF1	μF1	MF1
Aegis-Bench						
DCL	8.33	5.30	22.93	20.23	24.73	27.70
Qwen2.5-7B-SFT	5.05	2.80	60.03	22.70	19.61	16.90
VerifyMAS-7B-SFT	11.93 ↑6.88	5.95 ↑3.15	61.86 ↑1.83	22.96 ↑0.26	**29.95** ↑10.34	29.80 ↑12.9
Qwen3-8B-SFT	9.68	5.73	**64.79**	**38.96**	20.37	20.36
VerifyMAS-8B-SFT	**12.17** ↑2.49	**5.99** ↑0.26	60.19 ↓4.60	21.23 ↓17.73	29.94 ↑9.57	**30.34** ↑9.98
Who&When						
DCL	1.60	0.77	8.40	6.07	14.67	10.57
Qwen2.5-7B-SFT	1.26	0.52	43.51	32.51	6.77	4.20
VerifyMAS-7B-SFT	**5.25** ↑3.99	2.44 ↑1.92	42.00 ↓1.51	**33.69** ↑1.18	**14.85** ↑8.08	**13.25** ↑9.05
Qwen3-8B-SFT	5.17	2.33	**45.48**	30.77	8.00	5.29
VerifyMAS-8B-SFT	5.21 ↑0.04	**2.85** ↑0.52	40.09 ↓5.39	32.90 ↑2.13	13.35 ↑5.35	12.32 ↑7.03

Overall, the results suggest that VerifyMAS-SFT, trained on the constructed error hypothesis verification samples, can largely improve multi-agent failure attribution capability mainly by guiding the model to verify explicit error hypotheses over the full trajectory and localize the responsible agents for the entailed error. This design leads to stronger overall performance on failure attribution.

4.3 Fine-grained Failure Analysis

To better understand the behavior of different models, we conduct a fine-grained failure analysis by examining performance across individual error types and comparing with the DPR and CoT-Agent implemented based on Qwen2.5-7B-Instruct. Specifically, we report per-class Pair-F1 scores for each failure subtype and organize them into three groups—**Global**, **Local**, and **Hybrid** errors on Aegis-Bench [13]—to provide a fine-grained comparison of model performance across diverse failure patterns. The Error-F1 for each failure subtype across three methods can be found in the App. C.6. As shown in Fig. 3, VerifyMAS consistently improves error detection across all three categories compared with the DPR and CoT-Agent. The improvement is especially pronounced on Global and Hybrid Errors, showing that the proposed error-first and hypothesis-verification design is particularly effective at capturing failures that only become evident from the full interaction trajectory. At the same time, VerifyMAS also brings gains on Local errors, indicating that the method improves not only trajectory-level reasoning but also overall robustness in fine-grained failure attribution. A detailed case study is provided in the App. A.

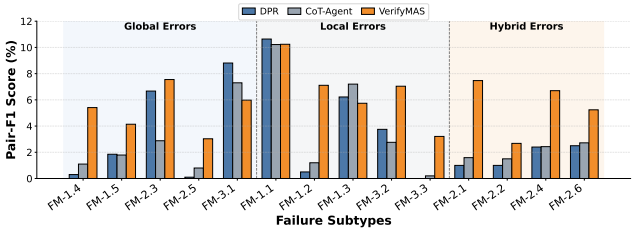

Figure 3: Per-class Pair-F1 score.

4.4 Generalization on Unseen Failures

To empirically demonstrate that VerifyMAS can learn a general evidence-grounded verification ability and can be adapted to new error types by introducing their corresponding natural-language hypotheses, we construct Aegis-Bench-U as an open-set evaluation dataset using the four held-out, previously unseen failure types. We hold out four failure types (FM-1.4, FM-1.5, FM-2.6, FM-3.3), selected as the last one or two types within each category, construct the training set using only the remaining ten types, and train VerifyMAS-7B-HO and Qwen2.5-7B-HO following the same procedure, where "-HO" denotes it is trained for attributing the held-out unseen failures. Specifically, during evaluation, each unseen failure type is represented by its corresponding natural-language hypothesis for VerifyMAS. Qwen2.5-7B continues to use the original AEGIS prompt template, augmented with definitions of the newly introduced failure types. The results are shown in Table 3. Note that Qwen2.5-7B-HO(SFT) performs worse than its zero-shot counterpart when encountering unseen errors on some specific metrics, while VerifyMAS-7B-HO(SFT) exhibits stronger generalization because it learns a general evidence–hypothesis verification mechanism rather than a fixed trajectory-to-label mapping. We also conduct an OOD dataset evaluation in this setting which can be found in App. C.4.

Table 3: Results of attributing unseen failures on an Aegis-Bench subset: Aegis-Bench-U.

Method	Pair		Error	
	μF1	MF1	μF1	MF1
Zero-shot				
Qwen2.5-7B	6.60	2.31	30.62	26.99
VerifyMAS	7.40	3.15	36.97	35.66
SFT				
Qwen2.5-7B-HO	3.99	1.49	12.89	9.53
VerifyMAS-7B-HO	**11.74**	**4.96**	**40.13**	**35.72**

4.5 Ablation Study

Hypothesis Verification vs. Plain Instruction. To assess whether the hypothesis testing provides additional benefits over the same paradigm, we design a **non-CoT error-first** attribution variant, named **Direct-Error**, which directly predicts the error from the trajectory and then identifies the corresponding agents. Then, we further introduce a **Chain-of-Thought**-based **Error-first** attribution variant, named **CoT-Error**, which follows the same error-first pipeline but explicitly generates intermediate reasoning before making the predictions. From Table 4, we note that both Direct-Error and CoT-Error fall behind VerifyMAS. The results indicate that direct error prediction alone is insufficient and highlight the necessity of formulating error hypotheses and verifying them against the full trajectory for reliable attribution in LLM-MAS.

Table 4: Ablation study.

Model	Aegis-Bench						Who&When						Avg.
	Pair		Agent		Error		Pair		Agent		Error		
	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1	μF1	MF1	
Direct-Error	4.20	0.86	37.32	10.75	12.24	7.74	0.68	0.47	36.55	30.02	0.69	0.48	11.83
CoT-Error	4.31	0.96	39.65	14.53	11.76	8.13	5.30	3.13	41.04	30.59	6.23	4.04	14.14
DPR	5.23	1.30	39.05	10.87	13.93	9.79	1.21	0.54	**44.65**	32.68	1.95	0.93	13.51
Direct-Agent	4.79	1.35	34.67	9.74	13.27	8.65	0.60	0.41	41.67	31.95	1.27	0.67	12.42
CoT-Agent	5.02	2.52	27.55	14.49	14.96	11.36	2.31	1.14	40.92	23.50	3.64	1.77	12.43
VerifyMAS	**5.89**	**2.98**	**47.92**	**14.66**	**25.07**	**24.26**	**5.83**	**3.44**	39.26	**33.50**	**13.33**	**9.54**	**18.81**

Error-first. vs. Agent-first. To evaluate the advantage of our error-first attribution design, in addition to **DPR** and **CoT-Agent**, we introduce a **non-CoT agent-first** attribution variant, named **Direct-Agent**. This variant first directly predicts the failed agent from the trajectory and then identifies the corresponding error type, serving as a contrast to **CoT-Agent**. As shown in Table 4, CoT-Agent slightly underperforms CoT-Error, indicating that incorporating the reasoning step within an error-first attribution process is more effective than starting from agents. Moreover, DPR and the two agent-first variants consistently yield inferior performance compared with VerifyMAS. This result further supports the effectiveness of the error-first attribution paradigm, as explicitly reasoning about failure modes before agent localization enables more reliable and accurate attribution.

4.6 Efficiency Analysis

We further evaluate the efficiency of our method by comparing the average processing time per sample with several existing local models. For a fair comparison, all the competing methods are implemented based on Qwen2.5-7B-Instruct and evaluated on Who&When [32]. As shown in Fig.4, VerifyMAS remains computationally efficient while delivering strong performance on the benchmark datasets. This indicates that VerifyMAS does not rely on excessive inference cost to achieve its gains, but instead offers a practical and scalable solution for effective failure attribution.

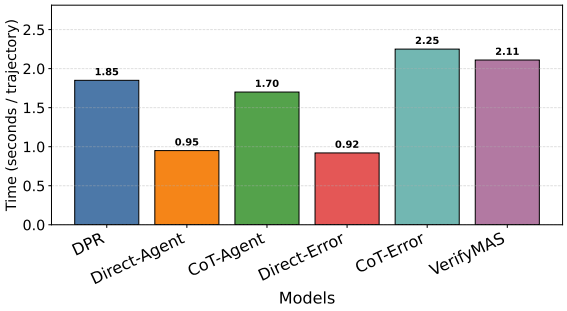

Figure 4: Running time comparison.

5 Conclusion

In this paper, we study automatic failure attribution in multi-agent systems and formulate it as an error-first hypothesis verification framework, VerifyMAS. It first verifies whether an error hypothesis is supported by the full interaction trajectory and then localizes the responsible agent, enabling the model to better capture global failures that depend on long-range context and provide more reliable and fine-grained trajectory failure analysis. We further developed a hypothesis verification-based data construction strategy and fine-tuned a specialized LLM verification model for trajectory-level failure verification and agent attribution. Experimental results on Aegis-Bench and Who&When demonstrate that VerifyMAS can consistently improve diverse open-source and proprietary models, and hypothesis-verification-based fine-tuning strengthens in-distribution diagnostic ability while preserving out-of-distribution generalization for failure attribution in LLM-MAS.

Limitation and Future Work. Our empirical results suggest that VerifyMAS improves failure attribution capability through its error-first paradigm and hypothesis verification design. Nevertheless, its verification-driven design may lead to less conservative predictions in certain trajectories. Future work will explore additional supplementary supervision to enable more precise failure localization.

Acknowledgment. This research was supported in part by the Agency for Science, Technology and Research (A*STAR) under its MTC Young Individual Research Grant (YIRG) Grant (M24N8c0103), the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 Grant (24-SIS-SMU-008) and the Lee Kong Chian Fellowship (T050273). Bing Liu's work was supported in part by U.S. National Science Foundation (NSF) grants IIS-2229876 and CNS-2225427.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 7

[2] Laksh Advani. Trajectory guard–a lightweight, sequence-aware model for real-time anomaly detection in agentic ai. *arXiv preprint arXiv:2601.00516*, 2026. 3

[3] Adi Banerjee, Anirudh Nair, and Tarik Borogovac. Where did it all go wrong? a hierarchical look into multi-agent error attribution. *arXiv preprint arXiv:2510.04886*, 2025. 2

[4] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025. 2, 3

[5] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 7

[6] Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025. 14, 16

[7] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024. 1

[8] Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. Aegis2. 0: A diverse ai safety dataset and risks taxonomy for alignment of llm guardrails. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5992–6026, 2025. 3

[9] Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *Advances in neural information processing systems*, 37:8093–8131, 2024. 3

[10] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 7

[11] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023. 3

[12] Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. Mas-fire: Fault injection and reliability evaluation for llm-based multi-agent systems. *arXiv preprint arXiv:2602.19843*, 2026. 3

[13] Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. *arXiv preprint arXiv:2509.14295*, 2025. 2, 3, 6, 7, 9, 14, 16

[14] Yuta Koreeda and Christopher D Manning. Contractnli: A dataset for document-level natural language inference for contracts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1907–1919, 2021. 2

[15] Ohjoon Kwon, Donghyeon Jeon, Nayoung Choi, Gyu-Hwung Cho, Hwiyeol Jo, Changbong Kim, Hyunwoo Lee, Inho Kang, Sun Kim, and Taiwoo Park. Slm as guardian: Pioneering ai safety with small language model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1333–1350, 2024. 3

[16] Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yuejin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qingyu Liu, et al. Agentdog: A diagnostic guardrail framework for ai agent safety and security. *arXiv preprint arXiv:2601.18491*, 2026. 1, 2, 3, 7, 8

[17] Junjun Pan, Yixin Liu, Rui Miao, Kaize Ding, Yu Zheng, Quoc Viet Hung Nguyen, Alan Wee-Chung Liew, and Shirui Pan. Explainable and fine-grained safeguarding of llm multi-agent systems via bi-level graph anomaly detection. *arXiv preprint arXiv:2512.18733*, 2025. 3

[18] Divya Pathak, Harshit Kumar, Anuska Roy, Felix George, Mudit Verma, and Pratibha Moogi. Detecting silent failures in multi-agentic ai trajectories. *arXiv preprint arXiv:2511.04032*, 2025. 1

[19] Hezhe Qiao, Hanghang Tong, Bo An, Irwin King, Charu Aggarwal, and Guansong Pang. Deep graph anomaly detection: A survey and new perspectives. *IEEE Transactions on Knowledge and Data Engineering*, 2025. 3

[20] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025. 7

[21] Qwen Team. Qwen3 technical report, 2025. 7

[22] Qwen Team. Qwen3. 5-omni technical report. *arXiv preprint arXiv:2604.15804*, 2026. 7

[23] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023. 3

[24] Megan Ung, Jing Xu, and Y-Lan Boureau. Saferdialogues: Taking feedback gracefully after conversational safety failures. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6462–6481, 2022. 1

[25] Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7261–7276, 2025. 3

[26] Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, et al. Guardagent: Safeguard llm agents via knowledge-enabled reasoning. In *International Conference on Machine Learning*, pages 68316–68342. PMLR, 2025. 3

[27] An Yang et al. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024. 7

[28] Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Proud, Piyush Kumar, Bhaktipriya Radharapu, et al. Shieldgemma: Generative ai content moderation based on gemma. *arXiv preprint arXiv:2407.21772*, 2024. 3

[29] Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the LLM agentic systems? *CoRR*, abs/2509.03312, 2025. 3

[30] Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *International Conference on Machine Learning*, pages 76678–76692. PMLR, 2025. 1

[31] Heng Zhang, Yuling Shi, Xiaodong Gu, Haochen You, Zijian Zhang, Lubin Gan, Yilei Yuan, and Jin Huang. Graphtracer: Graph-guided failure tracing in llm agents for robust multi-turn deep search. *arXiv preprint arXiv:2510.10581*, 2025. 2

[32] Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025. 2, 3, 6, 7, 10, 14

[33] Lifan Zheng, Jiawei Chen, Qinghong Yin, Jingyuan Zhang, Xinyi Zeng, and Yu Tian. Rethinking the reliability of multi-agent system: A perspective from byzantine fault tolerance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 35012–35020, 2026. 3

[34] Jialong Zhou, Lichao Wang, and Xiao Yang. Guardian: Safeguarding llm multi-agent collaborations with temporal graph modeling. *arXiv preprint arXiv:2505.19234*, 2025. 3

[35] Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025. 2, 3, 7, 8

[36] Mingchen Zhuge, Changsheng Zhao, Dylan R Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. In *International Conference on Machine Learning*, pages 80569–80611. PMLR, 2025. 3

[37] Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, et al. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025. 1

A Case Study

To provide a qualitative illustration of our model's diagnostic capabilities, we present a case study of trajectory failure attribution. As shown in Fig. 5, the MAS was tasked with solving a physical problem. This case aims to demonstrate the advantage of VerifyMAS in detecting global failures in multi-agent trajectories.

From the conversation history, we note the following

FM-2.4: Hiding or Distorting Important Information. Although the **Solver** initially produces the correct answer, later agents introduce several errors. The **Evaluator** incorrectly rejects the correct solution and introduces inconsistent numerical claims by claiming that they are not relevant, although they are exactly required by $E = F/q$. This hides the key evidence needed to solve the problem correctly, which distorts important information and corresponds to FM-2.4.

FM-1.4: Removed or Ignored Conversation History. The **Critic** further misreads the previous conversation by claiming that the chat history contains an incorrect answer, even though the earlier solution is correct, indicating FM-1.4. It ignores the earlier correct calculation of $1500\,\mathrm{N/C}$ and accepts the later inconsistent answer of $300\,\mathrm{N/C}$, showing that important previous context is not properly preserved.

FM-3.2: Skipped Verification. The **Solver** skips a basic numerical check. Substituting $F = 3.0 \times 10^{-6}\,\mathrm{N}$ and $q = -2.0 \times 10^{-9}\,\mathrm{C}$ gives an electric-field magnitude of $1500\,\mathrm{N/C}$, so the final answer $300\,\mathrm{N/C}$ should have been rejected. Instead of verifying this inconsistency, the **Solver** prematurely introduces additional roles like senior physicist, allowing the incorrect answer to remain unresolved.

While the direct prediction baseline only predicts FM-3.1 and misses the true failure modes, VerifyMAS successfully identifies the key ground-truth errors, including FM-1.4, FM-2.4, and FM-3.2. This example shows that VerifyMAS can better capture trajectory-level, context-dependent failures than direct prediction methods.

B Details of Datasets

The key statistics of the datasets are presented in Table 5. A detailed introduction of these datasets is given as follows.

- AEGIS [13]: AEGIS contains 9,533 trajectories with 24,843 errors collected from six task domains, including MATH, SciBench, GSM8K, HumanEval, MMLU, and GAIA, and six multi-agent system frameworks, including LLM Debate, MacNet, AgentVerse, DyLAN, SmolAgents, and Magentic-One. The dataset is further divided into three non-overlapping subsets. Following the AEGIS protocol, the test set, Aegis-Bench, consists of 100 trajectories sampled from each of the six benchmarks. The remaining data is split into training and validation sets with an 80%/20% ratio. All splits are generated using a fixed random seed to ensure reproducibility.
- Who&When [32]: Who&When is a failure logs dataset from 127 LLM multi-agent systems with fine-grained annotations linking failures to specific agents and decisive error steps. It contains 184 trajectories with 1,516 annotated errors. Since Who&When only provides free-text mistake explanations, it is preprocessed by using an LLM, Gemini-2.5-Flash, to map each explanation to one of the 14 predefined error modes, thereby ensuring consistent evaluation labels.
- TRAIL [6] : TRAIL contains 148 human-annotated agentic traces. Its annotations were conducted by four annotators with expertise in software engineering and log debugging, with inter-annotator agreement evaluated on an additional set of traces. TRAIL contains trajectories from two benchmarks, GAIA and SWE-Bench, generated using agent models newer than Qwen2.5-3B, thereby introducing both temporal and model distribution shifts relative to our training data. Since the failure annotations in TRAIL do not directly follow our 14-type error taxonomy, we used Gemini-2.5-Flash to map the original TRAIL failure descriptions to the corresponding VerifyMAS error types, following the annotation protocol in [13].

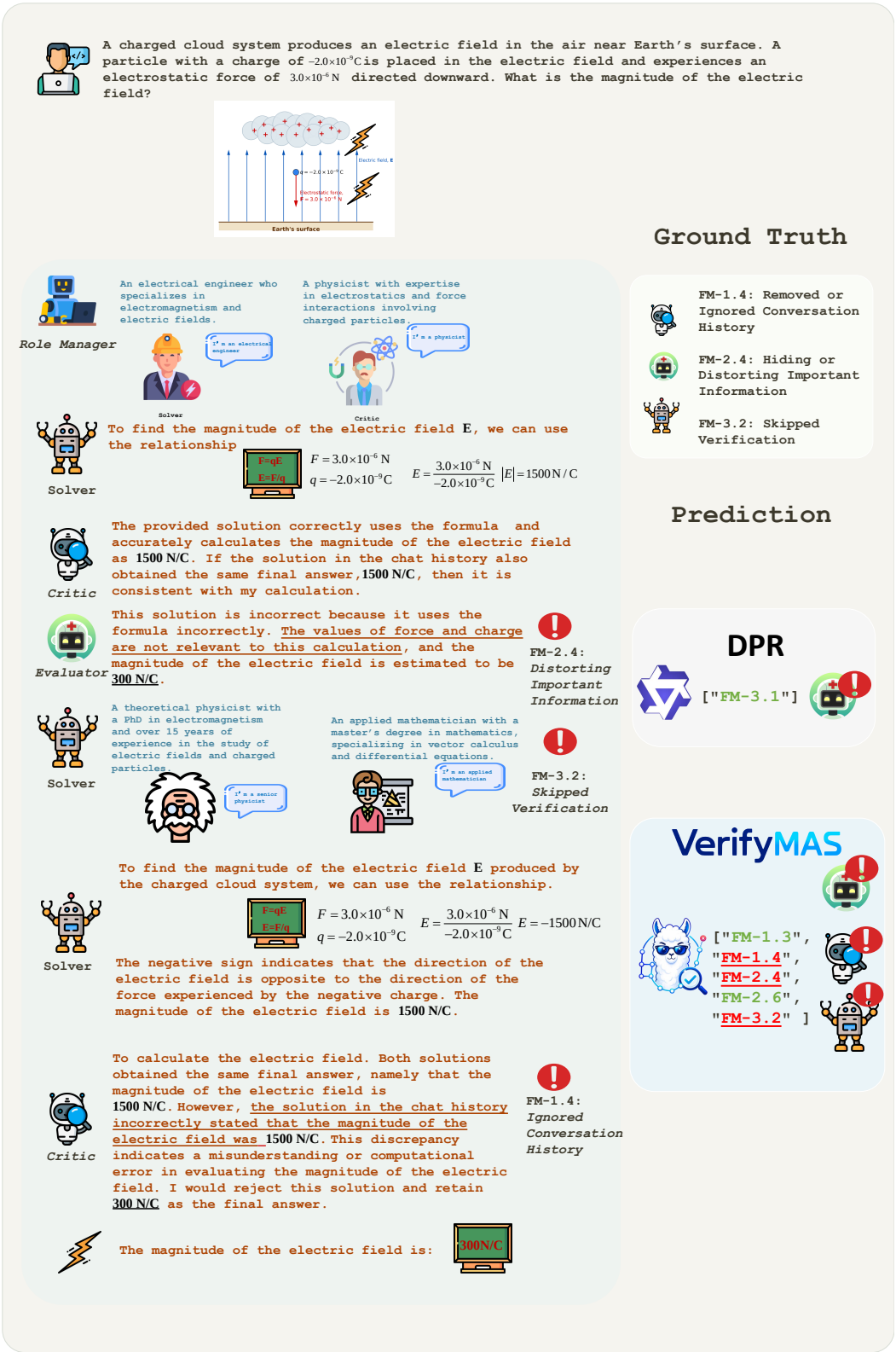

Figure 5: A case study of a trajectory failure attribution, DPR and VerifyMAS are based on Qwen2.5-7B-Instruct.

Table 5: Distribution of error types in Aegis-Bench, Who&When and TRAIL.

Error Type	Error Definition	Aegis-Bench	Who&When	TRAIL
FM-1.1	Deviates from task requirements.	2,310	145	338
FM-1.2	Acts outside the assigned role.	1,824	103	1
FM-1.3	Adds redundant steps.	1,626	92	23
FM-1.4	Ignores conversation history.	1,654	87	23
FM-1.5	Misses stopping criteria.	2,177	108	60
FM-2.1	Repeats completed tasks.	1,869	99	37
FM-2.2	Makes requests ambiguous.	1,758	132	6
FM-2.3	Drifts from the main goal.	1,823	110	48
FM-2.4	Hides key information.	1,713	104	73
FM-2.5	Ignores other agents' inputs.	1,660	100	49
FM-2.6	Uses inconsistent reasoning.	1,513	103	46
FM-3.1	Stops the task prematurely.	1,647	113	14
FM-3.2	Skips verification steps.	1,618	114	116
FM-3.3	Performs incorrect verification.	1,651	106	15
Total	–	24,843	1,516	849
Test Trajectory	–	600	184	148

Error Types. In AEGIS [13], agent failures are defined using a taxonomy of 14 fine-grained error types, covering task execution errors, communication and coordination errors, and quality and verification errors, as shown in Table 5. These categories describe different functional aspects of multi-agent failures, ranging from incorrect task understanding and execution, to flawed inter-agent communication, to missing or ineffective verification. In addition to this taxonomy-based categorization, we further analyze errors from another perspective according to the context required for detection. Specifically, we divide error modes into **Local**, **Global**, and **Hybrid** errors. As shown in Fig. 6, global errors require full-trajectory context, task goals, or cross-step dependencies to identify; Local errors can be detected mainly from a single agent's local behavior or immediate response; and Hybrid errors require both local behavioral evidence and global trajectory-level context. This grouping enables a more structured analysis of whether different methods are better at detecting local agent-level mistakes or context-dependent failures that emerge across the whole multi-agent trajectory.

C Additional Experimental Results

C.1 Additional Results on TRAIL

In this section, we added an evaluation on TRAIL [6], an independently constructed dataset rather than one generated by our own data pipeline. We report VerifyMAS results separately on both TRAIL benchmarks under zero-shot and SFT in Table 6. From the results, we observe that VerifyMAS outperforms the baseline on both zero-shot and SFT, demonstrating that VerifyMAS remains effective on unseen benchmarks comprising newer and independently collected trajectories. This showcases that the generalizability of VerifyMAS is not restricted by the base models or trajectory-generation process used in our original datasets. Moreover, the baseline Qwen2.5-7B (SFT) experiences a performance degradation compared to Qwen2.5-7B, because it is trained using a direct-mapping objective, which does not generalize well to unseen datasets. In contrast, VerifyMAS-7B (SFT) consistently outperforms its zero-shot counterpart, demonstrating that the proxy-task-based hypothesis-verification formulation enables the model to learn more transferable and generalizable attribution capabilities.

C.2 VerifyMAS vs. Per-agent/error Checking.

In this section, we introduced three structured variants of VerifyMAS, including a **per-error checklist**, a **per-agent checklist**, and **per-agent error-hypothesis verification**. For the checklist-based variants, we do not introduce the concept of a hypothesis. Instead, we explicitly ask whether a particular condition occurs. This formulation can be applied at both the error level and the agent level. Specifically,

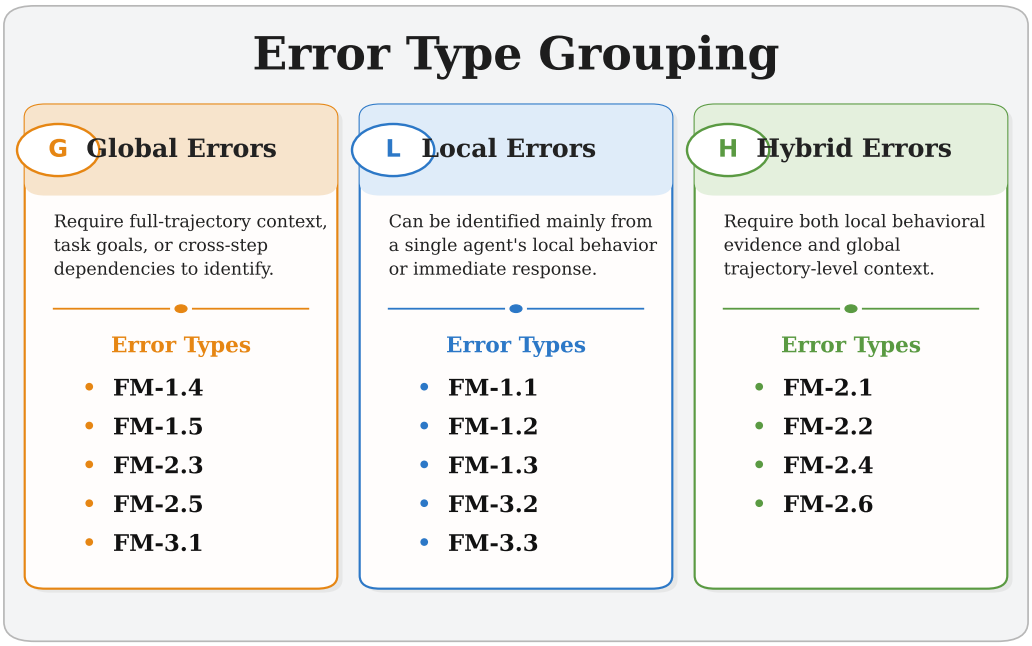

Figure 6: The category of error types.

Table 6: Performance gains and SFT performance of VerifyMAS on TRAIL.

Method	Pair		Agent		Error	
	μF1	MF1	μF1	MF1	μF1	MF1
Zero-shot						
Qwen2.5-7B	3.28	1.69	24.30	7.15	12.36	8.81
+VerifyMAS	7.58 $\uparrow$4.30	3.33 $\uparrow$1.64	30.31 $\uparrow$6.01	8.75 $\uparrow$1.60	24.91 $\uparrow$12.55	20.41 $\uparrow$11.60
GPT-4o-mini	1.16	0.81	3.25	1.46	**39.71**	32.72
+VerifyMAS	**12.51** $\uparrow$11.35	**8.87** $\uparrow$8.06	**87.41** $\uparrow$84.16	**51.70** $\uparrow$50.24	38.26 $\downarrow$1.45	**34.83** $\uparrow$2.11
SFT						
Qwen2.5-7B (SFT)	1.27	0.57	8.33	2.60	12.75	5.85
VerifyMAS-7B (SFT)	**7.79** $\uparrow$6.52	**3.92** $\uparrow$3.35	**31.81** $\uparrow$23.48	**9.95** $\uparrow$7.35	**37.83** $\uparrow$25.08	**34.84** $\uparrow$28.99

the per-error checklist asks whether each error type occurred, whereas the per-agent checklist asks whether a given agent made a mistake. For the per-agent error-hypothesis verification variant, we formulate the hypothesis around the agent rather than the error type. When the hypothesis that a particular agent made a mistake is supported, the method subsequently identifies the corresponding error type. The results are shown in Table 7.

From the results, we observe that (1) both per-agent verification and the per-agent checklist underperform VerifyMAS and the per-error checklist, further demonstrating the effectiveness of the error-first mechanism; (2) VerifyMAS outperforms the per-error checklist across all metrics except Pair MF1 on TRAIL, highlighting the importance of explicitly reformulating failure attribution around error hypotheses and leveraging these hypotheses as guided priors; and (3) VerifyMAS achieves the best overall performance among the newly added structural variants on both Aegis-Bench and TRAIL, demonstrating its strong capability for reliable and accurate failure attribution through its error-first design and hypothesis-verification mechanism.

C.3 Coarse Category Performance

In this section, we additionally evaluate the methods using three coarse-grained categories: Task-related failures, Communication and Coordination failures, and Quality and Verification failures. As shown in Table 8, the performance of VerifyMAS improves substantially under this coarser taxonomy,

Table 7: VerifyMAS vs. Per agent/error checking on Aegis-Bench and TRAIL.

Method	Pair μF1	Pair MF1	Agent μF1	Agent MF1	Error μF1	Error MF1
Aegis-Bench						
Per-error checklist	4.36	2.11	31.55	12.13	21.29	20.12
Per-agent checklist	3.74	1.15	38.18	12.09	14.62	9.88
Per-agent verification	3.22	0.95	32.27	11.66	9.34	6.07
VerifyMAS	**5.89**	**2.98**	**47.92**	**14.66**	**25.07**	**24.26**
TRAIL						
Per-error checklist	6.06	**4.69**	28.49	7.71	21.98	18.71
Per-agent checklist	2.25	1.51	29.41	**9.44**	11.06	8.28
Per-agent verification	1.51	0.85	27.04	8.27	5.06	3.35
VerifyMAS	**7.58**	3.33	**30.31**	8.75	**24.91**	**20.41**

Table 8: Coarse Category Results on Aegis-Bench.

Method	Pair μF1	Pair MF1	Error μF1	Error MF1
Qwen2.5-7B (Baseline)	16.56	4.47	46.34	43.89
VerifyMAS (Zero-shot)	24.02	7.59	68.26	67.37
VerifyMAS-7B (SFT)	**32.02**	**10.03**	**73.37**	**72.59**

indicating that much of the error in the fine-grained failure attribution comes from distinguishing between closely related subtypes rather than failing to recognize the general source of failure.

C.4 Generalization on Unseen Failures on OOD Dataset

To further demonstrate that SFT-based VerifyMAS models learn a general evidence–hypothesis compatibility function that applies to both seen and unseen failure hypotheses—and can adapt to new error types by adding their corresponding natural-language hypotheses, we evaluate unseen-error generalization on TRAIL-U, an OOD dataset. TRAIL-U is constructed following the same process in Sec.4.4. The results in Table 9 show that VerifyMAS(Zero-shot) can still effectively identify previously unseen error types on the OOD dataset, as demonstrated by its consistent improvements over Qwen2.5-7B(Zero-shot), the baseline adopted in AEGIS. Meanwhile, VerifyMAS-7B-HO(SFT) exhibits strong generalization on the OOD dataset when identifying the unseen failures. Consequently, VerifyMAS can be readily adapted to new error types and datasets by adding the corresponding natural-language hypotheses for newly encountered failures.

C.5 Precision and Recall Results

To complement the main results, this section provides additional details on model performance. Table 10 and Table 11 present a more fine-grained breakdown of Micro-Precision (μP) Macro Precision (MP), Micro Recall (μR) and Macro Recall (MR) results.

C.6 Fine-grained Failure Analysis

In this section, we report the error attribution performance only (Error-F1), without considering whether the faulty agent is correctly identified in Fig. 7 and 8. We observe that model performance varies significantly across error types. Errors related to clear task violations or explicit contradictions (e.g., role deviation or incorrect verification) are generally easier to detect while more subtle categories, such as incomplete reasoning, missing coordination, or implicit context neglect, remain challenging across all models. Overall VerifyMAS brings clear gains on three types of errors, indicating that the method improves not only trajectory-level reasoning but also overall robustness in fine-grained failure attribution.

Table 9: Results of attributing unseen failures on an TRAIL subset: TRAIL-U.

Method	Pair		Error	
	μF1	MF1	μF1	MF1
Zero-shot				
Qwen2.5-7B (baseline)	8.20	2.89	13.11	10.86
VerifyMAS	12.24	4.13	32.68	28.59
SFT				
Qwen2.5-7B-HO (baseline)	3.24	1.25	27.27	13.13
VerifyMAS-7B-HO	**21.43**	**4.37**	**51.94**	**46.41**

Table 10: Micro-Precision (μP) and Macro-Precision (MP) results of VerifyMAS-enabled models over their corresponding base models across three evaluation levels: Pair, Agent, and Error.

Model	Aegis-Bench						Who&When					
	Pair		Agent		Error		Pair		Agent		Error	
	μP	MP	μP	MP	μP	MP	μP	MP	μP	MP	μP	MP
Open-source Models												
Qwen2.5-7B-Instruct	6.13	2.92	41.59	21.91	20.90	19.82	2.45	1.29	52.10	25.86	4.11	3.16
+VerifyMAS	3.36	1.66	44.23	14.39	16.24	16.30	3.43	3.45	34.14	33.57	8.34	7.46
Qwen3-8B-Non-Thinking	4.78	2.00	28.51	10.23	20.60	16.77	4.68	2.20	34.36	20.66	5.15	2.79
+VerifyMAS	3.20	1.27	43.54	13.12	18.59	19.72	4.18	2.61	42.31	25.40	10.32	6.22
Qwen2.5-14B-Instruct	7.40	3.59	43.65	13.84	23.15	6.52	0.00	0.00	55.22	35.79	3.30	2.68
+VerifyMAS	4.22	2.02	49.08	15.26	18.75	18.65	3.50	2.72	36.07	29.21	9.78	3.19
Qwen3.5-9B-Non-Thinking	3.87	1.17	38.07	13.92	17.58	20.71	3.98	2.63	40.00	35.08	8.81	3.95
+VerifyMAS	3.17	1.51	41.09	14.90	17.50	19.63	3.39	2.83	36.19	35.48	9.14	7.91
Proprietary Models												
GPT-4o-mini	6.65	2.50	45.58	17.27	23.36	18.86	2.81	0.93	54.18	38.58	6.53	4.01
+VerifyMAS	3.54	1.59	40.25	14.79	16.64	16.67	2.75	3.01	30.59	36.83	6.92	7.31
GPT-4.1	9.26	3.22	45.68	13.43	24.62	18.62	4.95	1.68	47.53	33.72	9.20	7.33
+VerifyMAS	3.98	1.88	40.63	15.40	17.38	18.12	2.89	2.63	30.63	35.21	7.93	9.04
Gemini-2.5-Flash	8.91	3.94	51.26	18.89	28.50	23.11	8.99	3.76	66.34	41.50	14.95	9.88
+VerifyMAS	3.77	1.45	40.97	14.62	16.40	16.56	3.12	3.28	29.51	36.03	7.44	7.60
Gemini-2.5-Pro	9.09	3.93	49.93	18.54	24.54	20.54	8.31	3.67	64.01	38.58	14.31	9.79
+VerifyMAS	4.23	1.52	46.36	16.17	16.25	16.29	4.05	3.93	35.13	39.99	7.16	7.23

D Experiment Details

D.1 Training Data Construction Details

We construct the training data by converting each annotated trajectory into a set of hypothesis-verification instances.

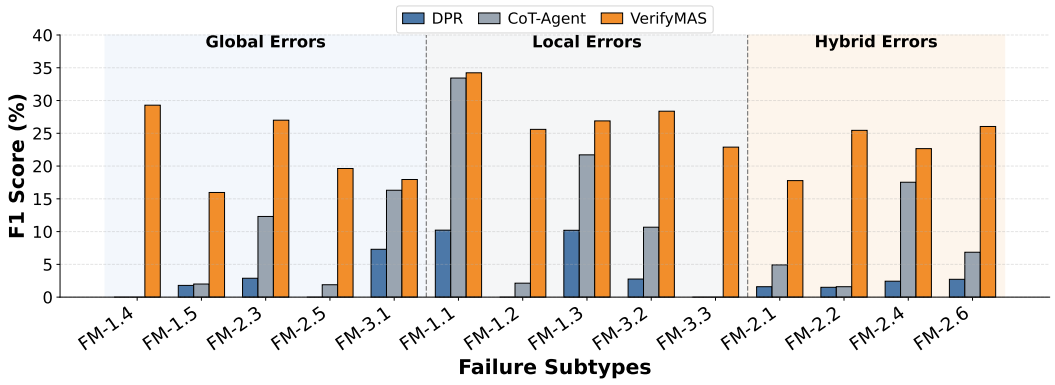

Figure 7: Fine-grained failure analysis grouped by the global, local, and hybrid errors on Aegis-Bench.

Table 11: Micro-Recall (μR) and Macro-Recall (MR) results of VerifyMAS-enabled models over their corresponding base models across three evaluation levels: Pair, Agent, and Error.

Model	Aegis-Bench						Who&When					
	Pair		Agent		Error		Pair		Agent		Error	
	μR	MR	μR	MR	μR	MR	μR	MR	μR	MR	μR	MR
Open-source Models												
Qwen2.5-7B-Instruct	4.26	3.20	20.6	13.17	11.65	10.63	2.17	1.08	33.7	22.71	3.26	1.35
+VerifyMAS	19.57	3.80	46.20	34.12	33.15	28.25	24.01	10.82	52.30	16.16	55.01	55.26
Qwen3-8B-Non-Thinking	2.94	0.71	15.61	6.28	13.36	9.96	2.63	1.24	21.4	12.98	2.87	1.23
+VerifyMAS	19.17	6.39	42.40	42.40	46.16	46.37	5.98	3.60	29.89	24.25	14.13	7.11
Qwen2.5-14B-Instruct	4.43	2.08	29.00	9.54	15.09	5.03	0	0	39.55	22.27	0.96	0.89
+VerifyMAS	29.83	10.81	48.83	14.96	58.30	57.95	8.82	8.74	35.87	27.40	13.04	10.73
Qwen3.5-9B-Non-Thinking	2.95	1.74	18.84	7.99	7.25	7.33	8.12	2.96	52.18	36.95	10.90	5.99
+VerifyMAS	23.77	10.17	42.29	17.56	59.97	61.41	32.69	3.80	65.77	38.60	50.00	34.60
Proprietary Models												
GPT-4o-mini	4.16	1.18	28.58	10.57	14.18	11.54	1.82	0.50	37.86	23.92	4.11	2.47
+VerifyMAS	47.54	17.83	63.60	21.98	86.17	86.15	46.20	3.94	84.78	40.83	59.24	39.56
GPT-4.1	5.48	1.38	27.71	8.63	14.79	10.99	2.56	0.80	30.28	20.37	5.56	3.98
+VerifyMAS	32.39	13.29	63.92	22.25	71.32	70.71	21.20	3.79	71.74	39.41	36.96	44.88
Gemini-2.5-Flash	5.1	2.16	30.93	11.77	17.43	14.95	6.13	2.42	41.48	26.23	8.55	5.83
+VerifyMAS	38.48	14.43	65.23	22.56	80.25	80.62	36.96	4.68	80.98	39.86	64.13	53.41
Gemini-2.5-Pro	4.82	2.17	29.73	11.65	14.90	12.76	5.23	1.86	38.51	24.15	7.82	5.58
+VerifyMAS	31.90	13.58	63.53	22.93	84.71	85.21	37.50	5.33	81.52	44.63	67.39	44.24

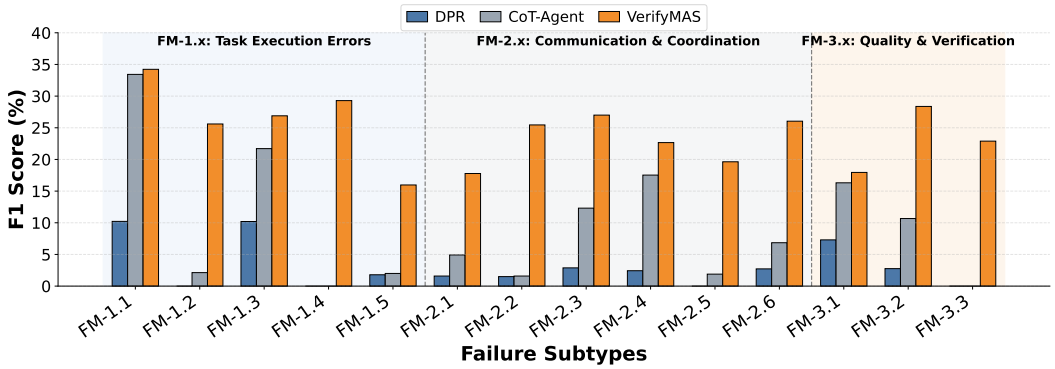

Figure 8: Fine-grained failure analysis grouped by task execution, communication& Coordination and quality & verification on Aegis-Bench.

Entail Sample Construction. For each trajectory, we construct positive entailment samples directly from the ground-truth failure annotations. Specifically, we first extract all annotated faulty agent–error pairs and group them by error type, since the same error hypothesis may be supported by multiple responsible agents. For each ground-truth error type, we instantiate a natural-language error hypothesis using the corresponding template from the predefined error dictionary. The resulting trajectory–hypothesis pair is labeled as entail, indicating that the trajectory provides evidence that this failure occurred. The target supervision contains all gold agents associated with this error type, enabling the model to learn both trajectory-level hypothesis verification and fine-grained multi-agent localization under the same positive hypothesis.

For example, if a given trajectory supports an error hypothesis of error type FM-3.2 and both the Planner and Solver are responsible for this failure, the target output is formatted as:

$$\mathbf{y} = \begin{matrix} \{\texttt{"label"} : \texttt{"A"}, \texttt{"agents"} : [\texttt{"Planner"}]\} \\ \{\texttt{"label"} : \texttt{"A"}, \texttt{"agents"} : [\texttt{"Solver"}]\}. \end{matrix}$$

For non-entailment samples, the agent set is empty, e.g.,

$$\{\texttt{"label"} : \texttt{"B"}, \texttt{"agents"} : []\}, \quad \{\texttt{"label"} : \texttt{"C"}, \texttt{"agents"} : []\}.$$

Heuristic Search for Negative Sampling. We use two complementary heuristics to construct negative hypotheses. First, counter-evidence heuristics search the trajectory for lexical and stage-level cues that explicitly refute an absent error hypothesis. Only absent hypotheses with sufficiently strong

Table 12: Counter-evidence keyword mapping for constructing contradicted examples. For each error type, we list representative lexical cues whose presence indicates that the trajectory is more likely to refute the corresponding error hypothesis.

Error Type	Counter-evidence Keywords
FM-1.1	`as requested; according to the requirement; followed the requirement; kept the required format; in python; met the specification`
FM-1.2	`according to assigned role; within role; as assigned; performed its role; stayed in its role`
FM-1.3	`without duplication; without redundant; concise; single final answer; did not repeat`
FM-1.4	`as mentioned earlier; based on previous; taking previous context into account; after the correction; considering earlier feedback; from previous turns`
FM-1.5	`base case; termination condition; stop when; stopping criterion; return when; until condition is met`
FM-2.1	`already completed so skip; no need to repeat; continue from previous result; reuse the previous result; avoid repeating`
FM-2.2	`clear instruction; explicit instruction; specifically asked; step-by-step instruction; clearly stated; explicitly stated`
FM-2.3	`focus on the main task; directly answer the query; stay on task; relevant to the task; aligned with the goal`
FM-2.4	`shared the important information; provided all relevant information; disclosed; made it clear; explicitly mentioned; fully informed`
FM-2.5	`incorporated feedback; addressed the concern; responded to the correction; followed the suggestion; took the feedback into account`
FM-2.6	`consistent with earlier; same conclusion as before; as stated earlier; remained consistent`
FM-3.1	`all requirements are satisfied; after verification; completed all requested parts; finished after checking; fully completed`
FM-3.2	`verified; validated; tested; checked; unit test; test case; cross-check`
FM-3.3	`verified against expected output; validated with test cases; cross-checked results; checked correctness; confirmed the result`

counter-evidence are labeled as *contradict*. The detailed counter-evidence for contradiction can be found in the Table 12.

Second, we define a nearby failure-type mapping to identify semantically related or easily confusable error types, as shown in Table 13. When explicit counter-evidence exists in the trajectory, nearby or absent failure hypotheses are prioritized as contradicted examples; otherwise, nearby hypotheses that are unsupported but not explicitly refuted are used as hard neutral negatives.

Oversampling for Rare Agents. During data construction, we also observe a long-tailed distribution over faulty agents. Some agents appear frequently in the training set, while others are associated with only a small number of positive attribution examples. Directly training on this imbalanced data may bias the model toward frequent agents and weaken its ability to identify rare but valid faulty agents. To alleviate this issue, we apply oversampling to rare-agent positive instances. Specifically, for each faulty agent, we count the number of entailment instances in which the agent is annotated as responsible. Agents with frequencies below a predefined threshold are treated as rare agents, and their corresponding positive instances are duplicated during training. This strategy increases the exposure of rare-agent attribution cases without changing the original label semantics. The ratio of each class for the constructed training dataset are shown in the Table. 14.

D.2 Training Details

The details of the hyperparameters are shown in the Table 15. We use the same training hyperparameters for fine-tuning both VerifyMAS-7B and VerifyMAS-8B models.

Table 13: Nearby error-type mapping for negative hypothesis construction. For each annotated error type, nearby error types denote semantically related or easily confusable absent errors.

Error Type	Nearby Error Types
FM-1.1	FM-2.3, FM-1.4, FM-1.2
FM-1.2	FM-1.1, FM-2.3, FM-2.5
FM-1.3	FM-2.1, FM-3.2, FM-2.3
FM-1.4	FM-2.5, FM-2.6, FM-1.1
FM-1.5	FM-3.1, FM-3.2, FM-3.3
FM-2.1	FM-1.3, FM-2.3, FM-2.5
FM-2.2	FM-2.4, FM-2.3, FM-2.5
FM-2.3	FM-1.1, FM-2.2, FM-2.1
FM-2.4	FM-2.2, FM-2.5, FM-1.4
FM-2.5	FM-1.4, FM-2.4, FM-2.6
FM-2.6	FM-1.4, FM-2.5, FM-3.3
FM-3.1	FM-1.5, FM-3.2, FM-3.3
FM-3.2	FM-3.3, FM-1.5, FM-3.1
FM-3.3	FM-3.2, FM-2.6, FM-3.1

Table 14: Distribution of three types of samples in the SFT training set.

Label	Type	# Samples	Ratio
A	Entail	17,753	53.02%
B	Neutral	5,767	17.23%
C	Contradict	9,961	29.75%
Total	–	**33,481**	**100.0%**

E Algorithm

The algorithms of zero-shot failure attribution and hypothesis verification-guided supervised fine-tuning are summarized in Algorithm 1 and Algorithm 2.

F Evaluation Prompts.

Here we list the prompts used when evaluating various open-source and closed-source models (divided into standard and CoT types). Since Aegis-Bench and Who&When have unified the data format, the same set of prompts is used for the two benches to ensure the fairness of the assessment. For completeness, we provide the prompts used for the baselines and ablation studies in the supplementary material.

Table 15: Training Hyperparameters for HSFT of VerifyMAS

Parameter	SFT Stage
Training Strategy	FSDP
Learning Rate	1e-4
Batch Size (Train)	64
Micro Batch Size per GPU	2
Max Sequence Length	8192
Total Epochs	2
LoRA Configuration	$r = 64$, $\alpha = 16$
Target Modules	all-linear
Gradient Clipping	1.0
Weight Decay	0.01
LR Scheduler	cosine
Warmup Steps Ratio	0.1
Model Precision	bf16
Gradient Checkpointing	True
CPU Offload	True (params)
Number of GPUs	2

Algorithm 1 Zero-shot Failure Attribution

Require: Trajectory τ_i, error type set $\mathcal{Y}$, candidate agent set $\mathcal{A}_i$, LLM verifier f_θ
Ensure: entail error list $\mathcal{Y}_i^*$ and selected responsible agents $\mathcal{A}_i^*$

$\mathcal{Y}_i^* \leftarrow \emptyset$, $\mathcal{A}_i^* \leftarrow \emptyset$
for each error type $y_m \in \mathcal{Y}$ **do**
  Construct hypothesis h_m from y_m
  Query f_θ with $(\tau_i, h_m, \mathcal{A}_i, P)$
  Obtain label $z_{i,m} \in \{\textsc{Entail}, \textsc{Neutral}, \textsc{Contradict}\}$
  if $z_{i,m} = \textsc{Entail}$ **then**
    $\mathcal{Y}^* \leftarrow \mathcal{Y}^* \cup \{y_m\}$
    Obtain responsible agents $\hat{\mathcal{A}}_{i,m} \subseteq \mathcal{A}$
    $\mathcal{A}_i^* \leftarrow \mathcal{A}_i^* \cup \{(y_m, a) \mid a \in \hat{\mathcal{A}}_{i,m}\}$
  end if
end for
return $\mathcal{Y}_i^*, \mathcal{A}_i^*$

Error Hypotheses

Type	Error Hypothesis
FM-1.1	An agent deviated from the specified task requirements in the trajectory.
FM-1.2	An agent acted outside its designated role in the trajectory.
FM-1.3	An agent introduced redundant or unnecessary steps in the trajectory.
FM-1.4	An agent overlooked important context from earlier conversation history in the trajectory.
FM-1.5	An agent failed to maintain a proper termination condition in the trajectory.
FM-2.1	An agent repeated a task that had already been handled in the trajectory.
FM-2.2	An agent made a request unclear or ambiguous for other agents in the trajectory.
FM-2.3	An agent drifted away from the main goal of the task in the trajectory.
FM-2.4	An agent omitted important information needed by other agents in the trajectory.
FM-2.5	An agent overlooked input or corrections from other agents in the trajectory.
FM-2.6	An agent used reasoning that was inconsistent with earlier statements in the trajectory.
FM-3.1	An agent ended the task before all requirements were met in the trajectory.
FM-3.2	An agent skipped necessary verification or validation steps in the trajectory.
FM-3.3	An agent performed flawed or unreliable verification in the trajectory.

Evaluation and SFT Prompt-VerifyMAS

You are a careful verifier for multi-agent trajectory failure attribution. You will be given: A trajectory; A failure hypothesis; The list of candidate agents appearing in the trajectory. Your task is to jointly predict:

- **label**: one of A, B, C
- **agents**: responsible agent(s) for the hypothesized failure

Label meanings: **A** = entail, **B** = neutral, **C** = contradict
Decision rules:

- Choose **A** when the trajectory provides clear or reasonably strong evidence that the hypothesized failure occurred, and that it negatively affected, or likely affected, the final outcome, final answer, final decision, or successful task completion.
- Choose **B** when the evidence is mixed, incomplete, weak, or the impact on the final outcome is uncertain.
- Choose **C** when the trajectory provides clear evidence that the hypothesis is false or inconsistent with what actually happened.
- If the failure appears to have been minor, corrected later, or not connected to the final outcome, prefer **B** over **A**.
- Use **C** only when the hypothesis is clearly contradicted by the trajectory, not merely because support is weak.

Agent attribution rules:

- A failure may be caused by one agent or multiple agents.
- If label = **A**, output one JSON object for each likely responsible agent.
- Each JSON object should contain exactly one responsible agent in the "agents" list.
- Include an agent only when there is clear or reasonably strong evidence that this agent contributed to the hypothesized failure.
- Do not include agents merely because they are mentioned near the error.
- Do not include agents that only followed instructions from another faulty agent unless their own action also contributed to the failure.
- Only output agent names that appear exactly in the provided candidate agent list.
- If label = **A** but no responsible agent can be confidently identified, output one JSON object with "agents": [].
- If label is **B** or **C**, output exactly one JSON object with "agents": [].

Output format:
Do not output thinking, reasoning, analysis, or `<think>...</think>`. Return JSON objects only. If there are multiple responsible agents, return multiple JSON objects, one per line.
Each JSON object must have exactly these keys:

```
{"label":"A","agents":["agent_name"]}
```

Valid examples:

```
{"label":"A","agents":["Planner"]}
{"label":"A","agents":["Solver"]}
{"label":"B","agents":[]}
{"label":"C","agents":[]}
```

Do not output explanations.
Do not output markdown.
Do not output extra text.

Algorithm 2 Hypothesis Verification-guided Supervised Fine-tuning

Input: Training trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, error taxonomy $\mathcal{Y}$, base LLM f_θ
Output: Fine-tuned model f_{θ^*}

Initialize SFT dataset $\mathcal{D}_{\text{sft}} \leftarrow \emptyset$
for each trajectory $\tau_i \in \mathcal{D}$ **do**
  Extract candidate agents $\mathcal{A}_i$ from trajectory τ_i
  Let $\mathcal{G}_i = \{(e, a)\}$ denote the ground-truth faulty agent-error pairs
  for each ground-truth pair $(e, a) \in \mathcal{G}_i$ **do**
    Construct an entailment hypothesis h_i^A for error type e
    Set target response $y_i^A = \{\texttt{"label"} : \texttt{"A"}, \texttt{"agent"} : [a]\}$
    Add $(\tau_i, h_i^A, \mathcal{A}_i, P, y_i^A)$ to $\mathcal{D}_{\text{sft}}$
  end for
  Sample contradiction error types $\mathcal{Y}_i^C \subseteq \mathcal{Y} \setminus \{e \mid (e, a) \in \mathcal{G}_i\}$
  Construct contradicted error types $\mathcal{Y}_i^C$ using explicit counter-evidence and the nearby failure-type mapping $\mathcal{M}_{\text{near}}$
  for each error type $y^C \in \mathcal{Y}_i^C$ **do**
    Construct a contradiction hypothesis h_i^C for y^C
    Set target response $y_i^C = \{\texttt{"label"} : \texttt{"C"}, \texttt{"agents"} : []\}$
    Add $(\tau_i, h_i^C, \mathcal{A}_i, P, y_i^C)$ to $\mathcal{D}_{\text{sft}}$
  end for
  Sample neutral error types $\mathcal{Y}_i^B$ from nearby but unsupported types in $\mathcal{M}_{\text{near}}$
  for each error type $y^B \in \mathcal{Y}_i^B$ **do**
    Construct a neutral hypothesis h_i^B for y^B
    Set target response $y_i^B = \{\texttt{"label"} : \texttt{"B"}, \texttt{"agents"} : []\}$
    Add $(\tau_i, h_i^B, \mathcal{A}_i, P, y_i^B)$ to $\mathcal{D}_{\text{sft}}$
  end for
end for
for each training step **do**
  Sample a mini-batch $\mathcal{B}$ from $\mathcal{D}_{\text{sft}}$
  Format each instance as input $\mathbf{x}_{i,m} = \{\tau_i, h_m, P, \mathcal{A}_i\}$ with trajectory corresponding to failure hypothesis h_m and target response $\mathbf{y}_{i,m}$
  Compute LM loss $\mathcal{L}_{\text{HSFT}} = -\frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x}_{i,m}, \mathbf{y}_{i,m}) \in \mathcal{B}} \sum_{t=1}^{|\mathbf{y}_{i,m}^*|} \log p_\theta \left(y_{i,m}^{(t)*} \mid \mathbf{x}_{i,m}, y_{i,m}^{(<t)*} \right)$
  Update model parameters θ by minimizing $\mathcal{L}_{\text{HSFT}}$
end for
return Fine-tuned model f_{θ^*}

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitation of our work in Conclusion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sec. 4 details implementation hyperparameters and dataset splits. The code is submitted in the supplementary materials.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code/data are submitted with the paper in supplemental material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental details, *e.g.*, Optimization learning rate, batch size, are presented in Sec. 4 and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: All the results reported in the paper are the mean over 5 runs. The performance is stable.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Experimental details like the memory and running time are provided in the main paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Our use of public datasets complies with NeurIPS Ethics Guidelines.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our model is not high-risk

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cited the data sources.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Not applicable (no new assets introduced in the paper)

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification:No human subjects or crowdsourcing was used.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Not applicable (no human subjects research).

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [No]

Justification: LLMs were only used for grammar checks (Grammarly). No impact on methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.