Spectrum-based Fault Localization for LLM-based Multi-Agent Systems

Identifying Faulty Agent Roles through Spectrum Analysis of Execution Traces

Le Kha Dan Nguyen[1]

Supervisor(s): Burcu Kulahcioglu Ozkan[1]**, Annibale Panichella**[1]**, Zahra Seyedghorban**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Le Kha Dan Nguyen
Final project course: CSE3000 Research Project
Thesis committee: Burcu Kulahcioglu Ozkan, Annibale Panichella, Zahra Seyedghorban, Matthijs Spaan

Abstract

Large Language Model-based Multi-Agent Systems (LLM-MAS) are promising frameworks for automating complex, real-world tasks. However, when these systems fail, failure attribution is challenging due to the stochastic behavior of Large Language Models (LLMs) and the distributed decision-making process of multi-agent collaboration. This paper investigates whether Spectrum-based Fault Localization (SBFL), a well-established technique in software testing and debugging, can be applied to identify faulty agent roles in LLM-MAS. We evaluate SBFL on HyperAgent across five SWE-bench Verified tasks, defining the spectra based on role message frequency and semantic output overlap. Agent roles are ranked by their computed suspiciousness scores and fault localization performance is measured using Top-1 and Top-3 accuracy against ground truth labels established by an LLM-as-a-judge. Our results show that semantic output overlap achieves the highest Top-1 accuracy of 60%, consistently outperforming raw message frequency spectra. However, none of the evaluated spectrum representations produces reliable Top-3 rankings, no SBFL formula consistently outperforms the others, and adding more execution runs does not consistently improve fault localization performance. These findings suggest that SBFL can support role-level fault localization in LLM-MAS, but its effectiveness depends strongly on spectrum design and remains limited for reliably identifying multiple faulty roles.

1 Introduction

Multi-Agent Systems (MAS) are increasingly explored for use in high-stakes domains, from supply chains and transportation networks to medical research and defense systems. The recent advent of Large Language Models (LLMs) has enabled the development of Large Language Model-based Multi-Agent Systems (LLM-MAS), in which multiple LLM-powered agents plan, use tools, and collaborate to complete tasks. LLM-MAS have attracted significant attention from the AI community [1], leading to the development of frameworks such as MetaGPT [2], ChatDev [3], and HyperAgent [4]. However, despite this growing interest, LLM-MAS have high failure rates, ranging from 41% to 86.7% across 7 state-of-the-art open-source frameworks [1]. Understanding where failures originate is therefore crucial for the safe deployment and systematic improvement of these systems.

Unlike traditional software systems, LLM-MAS exhibit probabilistic behavior and complex interactions between agents. As a result, failures may arise from multiple sources, including prompts, inter-agent communication, orchestration logic, or combinations of these, making them difficult to reproduce and isolate. Existing work on fault localization in LLM-MAS is limited in accuracy, as further discussed in Section 2.2.

Spectrum-Based Fault Localization (SBFL) is a well-studied technique in traditional software testing and debugging [5]. Crucially, SBFL requires no formal model of the system under investigation [6], instead deriving fault suspicion purely from statistical patterns across executions. This makes it especially well-suited for LLM-MAS, where probabilistic behavior makes any single execution unreliable as a diagnostic signal, yet systematic failure patterns may still emerge across repeated runs. Abreu et al. [7] showed that SBFL formulas which rank components by how frequently they appear in failing versus passing executions, achieve high localization accuracy on deterministic programs. Lee et al. [8] further demonstrated how fault localization accuracy improves when incorporating frequency execution count. Adapting these techniques to LLM-MAS, where components are probabilistic agent roles rather than deterministic program statements, is the main focus of this paper.

The choice of agent role as the unit of analysis is motivated by three considerations. First, agent roles form architectural boundaries in LLM-MAS frameworks. Systems such as MetaGPT [2], ChatDev [3], and HyperAgent [4] assign distinct responsibilities to each role, making them well-defined units whose behavior can be meaningfully attributed. Second, unlike individual messages or action steps, roles remain consistently identifiable across runs, enabling the aggregation of behavioral signals that SBFL requires. Third, role-level attribution is actionable. A faulty role can be traced back to the prompt, tool configuration, or coordination logic that defines it, which can then be revised to fix the failure. For these reasons, this paper treats agent roles as the analogue of program statements in traditional SBFL.

The paper aims to answer the following questions:

- **RQ1:** How can the core concepts in SBFL be mapped to agent roles in LLM-MAS execution traces?
- **RQ2:** What constitutes a suitable spectrum for agent roles in LLM-MAS?
- **RQ3:** To what extent can SBFL identify faulty agent roles, measured by Top-k accuracy against LLM-as-a-judge labels?

The main contribution of this paper is an empirical evaluation of SBFL as a fault localization technique for LLM-MAS, assessing how well this well-understood technique transfers to a stochastic agentic setting. For each task, a ranked suspiciousness list of agent roles is produced, then Top-1 and Top-3 accuracy is used to measure whether this ranking matches the ground truth ranking derived from LLM-as-a-judge labels [9], at the top position and top three positions, respectively.

The rest of this paper is organized as follows. Section 2 overviews SBFL and existing work on fault localization in LLM-MAS. Section 3 describes the methodology, including how the spectrum is defined, how suspiciousness scores are computed, and how performance is evaluated. Section 4 presents the experimental setup, results of the proposed methodology, and explicitly answers each research question. Section 5 discusses the results and addresses several threats to validity. Section 6 concludes the study and outlines directions for future work, while Section 7 states responsible research considerations, including reproducibility and integrity.

2 Related Work

2.1 Spectrum-based Fault Localization

Spectrum-based Fault Localization (SBFL) is a popular and lightweight automated diagnosis technique that identifies faulty program components through statistical correlations between system failures and the activity of the different parts of a system [6].

Concretely, SBFL operates on a coverage matrix together with a set of test outcomes. Let $S = \{s_1, \ldots, s_n\}$ denote the program components under analysis, such as statements or methods, and $T = \{t_1, \ldots, t_m\}$ denote the test cases, each labeled as passing or failing. Each entry $e_{s,t} \in \{0, 1\}$ indicates whether component s was executed during test t. From this matrix, four counts are derived for each component s:

- $a_{ef}(s)$: number of failing tests that execute s;
- $a_{ep}(s)$: number of passing tests that execute s;
- $a_{nf}(s)$: number of failing tests that do not execute s;
- $a_{np}(s)$: number of passing tests that do not execute s.

SBFL formulas combine these four counts into a suspiciousness score for each component. Components are ranked in descending order of their suspiciousness score and higher-ranked components are considered more likely to be responsible for the observed failures. Among the many formulas proposed in the literature, three of the most widely used are Tarantula [10], Ochiai [7], and DStar [11].

$$\text{Tarantula}(s) = \frac{\dfrac{a_{ef}(s)}{a_{ef}(s) + a_{nf}(s)}}{\dfrac{a_{ef}(s)}{a_{ef}(s) + a_{nf}(s)} + \dfrac{a_{ep}(s)}{a_{ep}(s) + a_{np}(s)}} \quad (1)$$

$$\text{Ochiai}(s) = \frac{a_{ef}(s)}{\sqrt{\big(a_{ef}(s) + a_{nf}(s)\big)\big(a_{ef}(s) + a_{ep}(s)\big)}} \quad (2)$$

$$\text{DStar}(s) = \frac{a_{ef}(s)^*}{a_{ep}(s) + a_{nf}(s)} \quad (3)$$

with $*$ typically set to 2, called Dstar2, to give disproportionately higher suspiciousness to components that are heavily exercised by failing tests.

It is worth noting that no formula has been shown to consistently outperform the others across all settings [5]. All three formulas share the same underlying principle that a component is more suspicious the more often it is exercised by failing tests and the less often it is exercised by passing tests, with the formulas differing primarily in how strongly each factor is weighted. Due to its lightweight and scalable nature, SBFL has become one of the most well-known fault localization techniques in software testing.

However, traditional SBFL assumes deterministic execution, binary coverage information, and well-defined test outcomes, assumptions that do not naturally hold for LLM-MAS. Consequently, applying SBFL to LLM-MAS for failure attribution requires several adaptations, as further discussed in Section 3.

2.2 Failure Attribution in LLM-based Multi-Agent Systems

As LLM-MAS become increasingly popular, so are the growing concerns regarding their reliability. MAST [1] provided the first systematic categorization of MAS failures, identifying 14 failure modes spanning 3 categories, namely system design issues, inter-agent misalignment, and task verification.

Acknowledging the same concerns and the labor-intensiveness of fault localizing LLM-MAS, Zhang et al. [12] introduced the first benchmark for automated failure attribution, which contains 184 annotated failure logs collected from 127 agentic systems. Their paper established several LLM-as-a-judge baselines, in which an LLM is used to evaluate execution traces as a scalable and automated alternative to manual annotation [9]. These included an *all-at-once* approach that analyzes the entire trajectory in a single pass, a *step-by-step* approach that reasons through actions sequentially, and a *binary-search* strategy that recursively narrows down the failure location. The benchmark demonstrated that even state-of-the-art reasoning models, including OpenAI o1 and DeepSeek-R1, the best method achieved only 53.5% accuracy in identifying the responsible agent and 14.2% accuracy in locating the decisive failure step. These findings indicate that failure attribution is a challenging and largely unsolved problem in LLM-MAS and motivate the need for more effective fault localization techniques.

Building upon this benchmark, Ge et al. [13] reformulated failure attribution as a SBFL problem over repeated trajectory replays. Their approach used a fine-grained action-triple spectrum, where each trajectory is decomposed into structured $\langle \text{agent}, \text{action}, \text{state} \rangle$ triples to enable alignment across executions. Based on this representation, they introduced a MAS-specific suspiciousness metric that combines agent-level behavior statistics, including coverage ratio and frequency proportion, with action-level signals such as local frequency enhancement and decay coefficients. Experimental results demonstrated substantial improvements over the results by Zhang et al. [12], achieving 29.35% action-level accuracy and outperforming LLM-based methods by 49%. However, FAMAS relies on fine-grained action-triple representations and MAS-specific suspiciousness metrics, leaving open the question of whether simpler and more readily available behavioral signals are sufficient for localizing faulty agent roles in LLM-MAS.

Together, these works represent the initial attempts at automated failure attribution for LLM-MAS. Whether effective failure localization can be achieved using lightweight signals and SBFL techniques remains largely unexplored.

3 Methodology

This section presents our adaptation of SBFL to LLM-MAS for identifying faulty agent roles. We begin by introducing the target LLM-MAS and benchmark tasks used to collect execution traces in Section 3.1, through which we construct three role-level spectrum representations in Section 3.2. Section 3.3 and Section 3.4 describe how suspiciousness scores are computed, how ground truth faulty agent roles are established, and how localization accuracy is evaluated.

3.1 System Under Study and Benchmark Tasks

We chose HyperAgent [4] as the target LLM-MAS for trace collection. It is designed for general software engineering tasks and has four primary roles, i.e., Planner, Navigator, Editor, Executor, each with distinct responsibility and produce attributable activity in execution traces. Since HyperAgent's role-based architecture reflects a common pattern across LLM-MAS frameworks, we do not consider the choice of HyperAgent critical to our approach and expect our findings to generalize well to other systems of this kind.

We then carefully selected five benchmark tasks from SWE-bench Verified, a widely used benchmark of real GitHub issues requiring code changes to resolve [14]. Tasks are chosen such that HyperAgent produces a roughly balanced mix of passing and failing runs across repeated executions, which is a prerequisite for reliable SBFL application. Our main motivation for choosing SWE-bench is that passing or failing task outcomes are automatically determined by the ground truth test suite, eliminating the need for manual labeling and potential errors arising from manual annotations.

3.2 Spectrum Definitions

As discussed in Section 2.1, traditional SBFL defines the spectrum as a binary matrix where each entry $e_{s,t} \in \{0, 1\}$ indicates whether component s is executed in test t. A direct adaptation to LLM-MAS is to retain this binary formulation, with components redefined as agent roles and each entry indicating whether a role participates in a given run.

However, many existing LLM-MAS frameworks, including MetaGPT [2], ChatDev [3], and HyperAgent [4], follow a fixed execution workflow in which nearly all roles participate in every run by design. Under a binary spectrum, such roles would therefore receive the same entry value across both passing and failing runs, leading to identical suspiciousness scores and tied ranking.

Studies on SBFL proposes resolving ties by incorporating frequency information [5] [8]. Taking this idea, we consider role-level spectrum representations that replace each binary entry with a weight reflecting how actively a role participates in a given run. This weighted formulation helps preserving the role-level abstraction while providing additional discriminative information for fault localization.

Formally, let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of agent roles in HyperAgent and let $T = \{t_1, t_2, \ldots, t_m\}$ be the set of collected runs for a given benchmark task. For each role r and run t, a spectrum value $k_{r,t}$ is extracted from the execution log, with its definition varying across the chosen spectrum representations. In this paper, the following three spectrum representations are evaluated:

Spectrum 1. Full-Trace Message Frequency The first spectrum defines $k_{r,t}$ as the total number of messages produced by role r over the full execution trace of run t. The intuition is that a faulty role may exhibit anomalous communication patterns, reflected in systematically different message volumes across failing and passing runs. This provides a simple yet informative signal, and is analogous to the execution frequency signal introduced by Lee et al. [8].

Spectrum 2. Message Frequency over the Final X% of the Trace Leading from Spectrum 1, Spectrum 2 restricts the analysis to the final $\lceil X\% \cdot L \rceil$ of an execution trace of length L, where $k_{r,t}$ denotes the number of messages produced by role r within this terminal segment. This design is inspired by TailTracer [15], which retains only the tail of a program execution trace based on the observation that software errors typically occur shortly before program crashes. By focusing on the tail segment of the trace, Spectrum 2 aims to reduce noise from earlier interactions that may be only weakly related to the eventual outcome.

Spectrum 3. Output Overlap The third spectrum captures semantic alignment between agent communications and the final code artifact. This addresses a limitation of raw message frequency, which does not distinguish between coordination-oriented communication and implementation-relevant messages. Inspired by LogRobust [16], which utilizes semantic representations to identify meaningful patterns in execution data, Spectrum 3 focuses on messages whose content is reflected in the final artifact. For each run t, $k_{r,t}$ is defined as the number of messages whose cosine similarity with the final code artifact exceeds a threshold τ.

The exact values of X and τ are treated as hyperparameters and their effects are examined empirically in Section 4.3.

3.3 Suspiciousness Scores and Rankings

For each role r and run t, the spectrum value $k_{r,t}$ defined in Section 3.2 gets mapped to a real value in $[0, 1]$ using the adapted sigmoid function proposed by Lee et al. [8]:

$$\mathrm{M}(k_{r,t}) = \begin{cases} \frac{1}{1+e^{-\alpha \cdot k_{r,t}}} & \text{if } k_{r,t} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where α is a constant that controls the steepness of the mapping. By default, α is set to 10, as it has been shown to improve fault localization performance across programs without prior knowledge of the number of bugs [8]. The resulting spectrum matrix contains real-valued entries $M(k_{r,t}) \in [0, 1]$ rather than binary, preserving information about intensity of each role's involvement in each run.

The mapped spectrum values are then aggregated into four counts for each role:

- a_{ef}: sum of $M(k_{r,t})$ over all failing runs;
- a_{ep}: sum of $M(k_{r,t})$ over all passing runs;
- a_{nf}: sum of $1 - M(k_{r,t})$ over all failing runs;
- a_{np}: sum of $1 - M(k_{r,t})$ over all passing runs.

These are then substituted directly into the three SBFL formulas mentioned in Section 2.1 to compute a suspiciousness score for each role, and roles are ranked in descending order of the resulting scores.

Motivating Example. Consider a typical HyperAgent execution on a SWE-bench Verified task. The PLANNER receives a GitHub issue and decomposes it into a sequence of subtasks, which it delegates to the NAVIGATOR. The NAVIGATOR explores the codebase to locate the files relevant to each subtask and passes this information to the EDITOR, who proposes code changes. The EXECUTOR then validates those changes by running the test suite and reports the outcome back to the PLANNER. Each primary role, except for the

PLANNER is further supported by an INNER sub-role and an INTERPRETER sub-role. Failures can propagate silently across role boundaries. For example, if the NAVIGATOR identifies the wrong files within the codebase, the EDITOR produces a patch for an irrelevant location, then the EXECUTOR's tests will fail. We use a single SWE-bench Verified task executed 20 times, denoted Task A, of which 9 runs pass and 11 runs fail, as a running example throughout this section. Table 1 presents $k_{r,t}$ across all 20 runs under Spectrum 1, together with the aggregated SBFL counts and resulting Ochiai suspiciousness scores. INNER-NAVIGATOR-ASSISTANT and NAVIGATOR INTERPRETER receive the highest suspiciousness scores (0.741 and 0.740), indicating that they are most strongly associated with failing executions, followed by NAVIGATOR (0.730) and PLANNER (0.722).

3.4 Ground Truth Establishment and Evaluation

Since no faults have been injected into HyperAgent and failing executions arise naturally during task execution, no ground truth faulty role(s) is directly available per failing run. To establish one, we use an LLM-as-a-judge evaluator [9]. Concretely, the complete execution trace of each failing run is provided to the evaluator, which is prompted to analyze the reasoning process, interactions, and outputs of all participating agent roles in order to identify which role or roles are most likely responsible for the task failure. We treat the resulting LLM-as-a-judge outputs as proxy ground truth labels, and defer discussion of their limitations to Section 5.

However, while SBFL produces a single ranking computed across all runs of a task, the LLM-as-a-judge labels each failing run individually. As a result, the two outputs are not directly comparable. To address this, let F denote the set of failing runs for a given task. For each role r and run $t \in F$, let faulty$(t) \subseteq R$ denote the set of roles identified as faulty in run t. LLM-as-a-judge outputs are aggregated by computing the proportion of failing runs in which each role is identified as faulty using the formula:

$$G(r) = \frac{1}{|F|} \sum_{t \in F} \mathbf{1}[r \in \text{faulty}(t)]$$

where $\mathbf{1}[\cdot] \in \{0, 1\}$ is the Iverson bracket. Roles are then ranked in descending order of $G(r)$, yielding a ranking comparable to the SBFL suspiciousness ranking.

Localization accuracy is evaluated using Top-1 and Top-3 accuracy. Top-1 accuracy is 1 if the role with the highest SBFL suspiciousness score matches the role with the highest $G(r)$ value, and 0 otherwise. Top-3 accuracy is 1 if the set of the three highest-ranked roles in the SBFL ranking exactly matches the set of the three highest-ranked roles according to $G(r)$, and 0 otherwise. In both cases, ties are resolved by ordering roles alphabetically.

Motivating Example (cont.) Returning to Task A, Table 2 shows the LLM-as-a-judge outputs across 11 failing runs of Task A while Table 3 aggregates these outputs. INNER-NAVIGATOR-ASSISTANT is identified most frequently, with $G(r) = 5/11$, followed by EDITOR with $G(r) = 3/11$, while INNER-EXECUTOR-ASSISTANT, EXECUTOR, and INNER-EDITOR-ASSISTANT each receive $G(r) = 1/11$, resulting in INNER-NAVIGATOR-ASSISTANT being ranked highest. Some runs identify multiple roles as responsible, reflecting that failures in LLM-MAS may involve several agents rather than a single source. This ranking matches the top position in the SBFL suspiciousness ranking from Table 1, where INNER-NAVIGATOR-ASSISTANT also ranks first. The Top-1 accuracy for Task A under Spectrum 1 is therefore 1. However, the Top-3 accuracy is 0 as the second and third positions in the SBFL ranking are NAVIGATOR INTERPRETER and NAVIGATOR, whereas the ground truth places EDITOR and INNER-EXECUTOR-ASSISTANT in those positions.

4 Experimental Setup and Results

4.1 Experimental Setup

Dataset Generation To collect execution traces for our experiments, we ran HyperAgent on five benchmark tasks using DeepSeek-V4-Flash as the underlying LLM for all agent roles. Each task was executed independently 20 times, resulting in a total of 100 runs. Figure 1 summarizes the number of passing and failing runs recorded per task. The only modification made to the original HyperAgent codebase was adapting its LLM backend to support DeepSeek, as the original implementation targets Azure OpenAI or local model deployments.

Execution logs are stored as plain-text `run.log` files, where each agent interaction is labeled with a speaker role and separated by an 80-character delimiter. The dataset is made publicly available.[1]

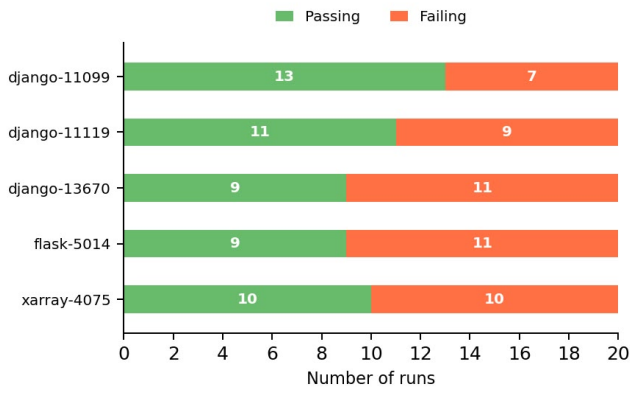

Figure 1: Run outcomes per benchmark task.

LLM-as-a-judge Pipeline Since stronger reasoning models do not necessarily outperform standard models in failure attribution, and prompt design is the dominant performance factor [12], we use DeepSeek-V4-Flash as our judge model. This choice is further motivated by cost efficiency at scale, as ground truth construction requires running the judge over many failing trajectories. Concretely, the model receives an alphabetically-ordered list of all agent roles participating in a failing run alongside the complete conversation transcript, and is prompted to identify which role(s) made the critical mistake(s) leading to failure. It returns its answer in JSON

[1] https://huggingface.co/datasets/lkdnguyen/traces

Table 1: Spectrum values and SBFL scores for Task A under Spectrum 1. The table shows per-run weighted participation, aggregated SBFL counts, and resulting Ochiai suspiciousness scores for each role.

Role	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20	a_{np}	a_{nf}	a_{ep}	a_{ef}	Score
Inner-Navigator-Assistant	3	3	3	17	29	12	10	24	23	9	14	15	6	5	4	4	5	5	7	4	0.146684	0.073191	8.853316	10.926809	0.740768
Navigator Interpreter	2	2	2	16	27	10	8	23	16	3	11	13	4	4	3	3	4	4	6	2	0.387235	0.187804	8.612765	10.812196	0.739609
Navigator	2	2	2	6	10	6	4	8	14	2	4	6	6	4	2	2	2	2	2	2	0.738149	0.518111	8.261851	10.481889	0.729987
Planner	4	5	4	9	6	6	4	5	10	8	6	6	5	4	3	1	2	1	2	8	0.129207	0.445089	8.870793	10.554911	0.722054
Inner-Executor-Assistant	0	8	5	8	0	14	6	0	0	16	20	7	7	7	7	0	0	0	0	13	2.010339	5.004055	6.989661	5.995945	0.501684
Executor Interpreter	0	7	4	6	0	14	5	0	0	14	19	7	7	7	7	0	0	0	0	12	2.021637	5.010989	6.978363	5.989011	0.501456
Executor	0	4	2	4	0	4	2	0	0	6	6	2	2	2	2	0	0	0	0	4	2.616473	5.294837	6.383527	5.705163	0.494746
Editor	4	2	2	6	0	0	0	0	4	0	2	2	2	2	4	1	0	0	8	3.741135	7.258865	8.147258	0.327149	0.327149
Inner-Editor-Assistant	5	2	3	17	0	0	0	0	7	0	6	2	7	6	3	5	7	5	7	10	0.188389	7.167540	8.811611	3.832460	0.324966
Editor Interpreter	3	1	2	15	0	0	0	0	5	0	5	1	5	5	2	4	4	1	2	6	0.477019	7.394838	8.522981	3.605162	0.312127
Outcome	P	F	P	F	F	F	F	F	F	F	P	P	P	P	P	F	P	F	F	P					

Table 2: LLM-as-a-judge identified faulty roles per failing run for Task A.

Run	Faulty Role(s)
run_02	Editor; Executor
run_04	Editor; Planner
run_05	Inner-Navigator-Assistant; Navigator Interpreter
run_06	Inner-Navigator-Assistant; Inner-Executor-Assistant
run_07	Executor; Planner
run_08	Inner-Navigator-Assistant
run_09	Editor; Navigator
run_10	Inner-Executor-Assistant; Executor Interpreter
run_16	Inner-Editor-Assistant; Inner-Executor-Assistant
run_18	Inner-Navigator-Assistant
run_19	Inner-Navigator-Assistant; Navigator Interpreter

Table 3: Ground truth ranking for Task A, aggregated from LLM-as-a-judge outputs. Each count reflects how many failing runs the evaluator identified that role as responsible for the failure.

Role	Count
Inner-Navigator-Assistant	5
Editor	3
Inner-Executor-Assistant	1
Executor	1
Inner-Editor-Assistant	1

format as {"faulty_roles": ["<role>", ...]}. The system and user prompts are publicly available and can be found in `judge.py` at our public repository [2]. The aggregated ground truth ranking is then computed as described in Section 3.4.

Spectrum Computation All three spectrum representations are extracted from `run.log` files. To select the optimal threshold, X is evaluated over $\{0.1, 0.2, 0.3, 0.5, 0.7, 1.0\}$ for Spectrum 2 and τ over $\{0.3, 0.5, 0.7, 0.9\}$ for Spectrum 3. For Spectrum 3, each role message and each task's final code artifact are embedded using OpenAI's `text-embedding-3-small` model and cosine similarity is computed to determine output overlap.

[2]https://github.com/nguyenlekhadan/role-based-SBFL

4.2 Fault Localization Accuracy

Table 4 reports Top-1 and Top-3 accuracy for each spectrum representation and SBFL formula under the best-performing hyperparameter configuration. For Spectrum 2, the optimal configuration is $X{=}0.1$ with the Tarantula formula. For Spectrum 3, all evaluated values of τ yield identical results and we therefore use $\tau{=}0.3$ as the representative.

Overall, Spectrum 3 achieves the highest Top-1 accuracy of 60% by correctly identifying the most suspicious role in three of the five tasks. Spectra 1 and 2 record a Top-1 accuracy of at most 40%, while the remaining configurations correctly identify the most suspicious role in only one of the five tasks, corresponding to a Top-1 accuracy of 20%.

However, Spectrum 3's Top-3 accuracy is 0% across all tasks, τ, and SBFL formulas. Non-zero Top-3 accuracy is only recorded by Spectrum 1 on `flask-5014` and Spectrum 2 at Tarantula, $X{=}0.1$ on `django-13670`. These two isolated instances are more likely coincidental than indicative of any systematic fault localization ability.

Notably, the choice of SBFL formula has negligible impact on fault localization performance. Tarantula, Ochiai, and DStar2 yield identical scores under Spectrum 3 for every task and threshold. Under Spectra 1 and 2, the formula affects individual task outcomes. For example, Tarantula correctly identifies the top role in `flask-5014` while Ochiai and DStar2 do not, yet this advantage does not generalize across tasks. Averaged across tasks, no formula consistently outperforms others, mirroring findings in traditional SBFL where no single technique claims to outperform all others under every scenario [5].

4.3 Sensitivity Analysis

Here, we examine the sensitivity of localization accuracy to three factors: the tail fraction X in Spectrum 2, the cosine threshold τ in Spectrum 3, and the number of execution runs considered per task.

Spectrum 2: Tail fraction X. Figure 2 reports average Top-1 and Top-3 across tasks for each value of X. The highest localization accuracy is achieved at $X{=}0.1$ under Tarantula (Top-1 = 0.4), however, given the small evaluation set of five tasks, this result is unlikely to be statistically meaningful. Ochiai and DStar2 show no corresponding improvement at $X{=}0.1$ and across all formulas, Top-1 accuracy fluctuates without a consistent trend as X increases. Overall, the benefits of trace truncation are inconclusive at this scale, as no value of X yields a consistent improvement.

Table 4: Top-1 and Top-3 accuracy per task across spectra and formulas. S1 = full-trace frequency; S2 = tail frequency (X=0.1); S3 = output overlap (τ=0.3). Tar = Tarantula, Och = Ochiai, DS = DStar2.

	S1						**S2** (X=0.1)						**S3** (τ=0.3)					
	Tar		Och		DS		Tar		Och		DS		Tar		Och		DS	
Task	T1	T3	T1	T3	T1	T3	T1	T3	T1	T3	T1	T3	T1	T3	T1	T3	T1	T3
django-11099	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0
django-11119	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**	0	**1**	0
django-13670	0	0	0	0	0	0	**1**	**1**	0	0	0	0	**1**	0	**1**	0	**1**	0
flask-5014	**1**	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
xarray-4075	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
Average	.2	.2	.2	.2	.2	.2	.4	.2	.2	.0	.2	.0	**.6**	.0	**.6**	.0	**.6**	.0

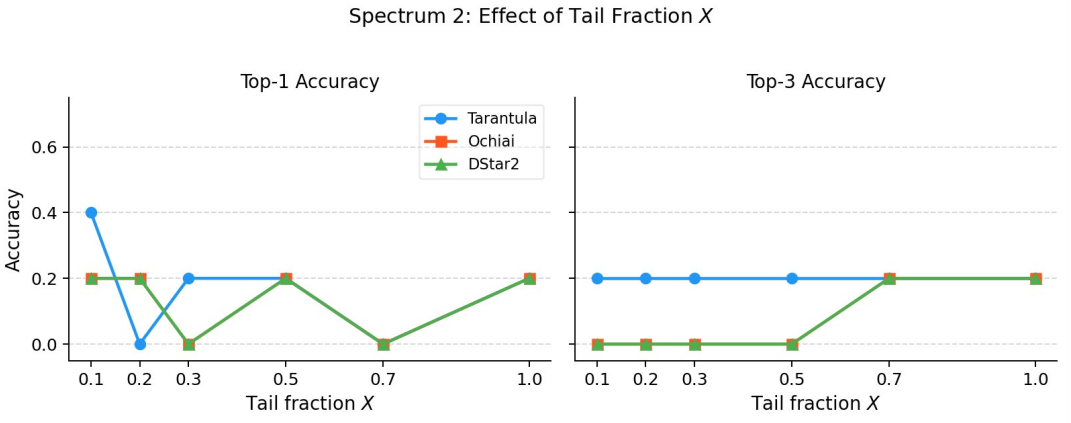

Figure 2: Spectrum 2 accuracy averaged over five tasks for varying fraction X and SBFL formula.

Spectrum 3: Cosine threshold τ. Figure 3 shows that Top-1 and Top-3 accuracy is insensitive to the choice of τ, remaining at 0.6 and 0, respectively, for all four threshold values and all three formulas. This indicates that role ranking by output overlap is stable regardless of where the binary threshold is drawn. A role whose communications align strongly with the final code artifact does so distinctly enough that varying the cutoff does not alter the resulting suspiciousness ranking. Consequently, τ need not be tuned in practice for this spectrum definition.

Number of Execution Runs Figure 4 reports how varying the number of runs available per task affects localization accuracy on Spectrum 1. Contrary to the expectation that additional runs improve localization accuracy, Ochiai and DStar2 achieve higher Top-1 (0.4) with 10 and 15 runs than with the full set of 20 runs. Given our small scale evaluation, no strong conclusions can be drawn. One possible explanation is that the LLM-as-a-judge ground truth becomes noisier as more runs are considered, with the judge attributing blame to different roles across runs, thereby destabilizing the top-1 ground truth role. This finding further suggests that ground truth quality, rather than spectrum representation, may be the limiting factor to our reported fault localization performance and motivates the use of more structured, human-validated annotations in future work on a larger dataset.

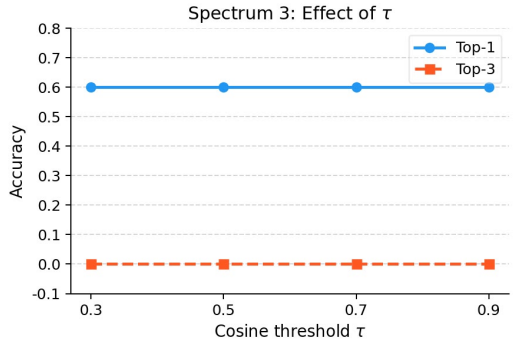

Figure 3: Spectrum 3 accuracy averaged over five tasks for varying cosine threshold τ and SBFL formula.

4.4 Summary of Evaluations

We now bring together the SBFL concept mapping from Section 3 and the empirical results from Sections 4.2 and 4.3 to answer each research question:

RQ1: SBFL Concept Mapping. Traditional SBFL, as described in Section 2.1, is built around four core concepts: program components, test cases, test outcomes, and a coverage matrix. Applying SBFL to localize faulty agent roles in LLM-MAS therefore requires identifying an equivalent of each

concept at the role level. In this paper, we consider agent roles as the analogue of program components, individual executions of the LLM-MAS as the analogue of test cases, and the success or failure of an execution as the test outcome. While the first three concepts map naturally onto LLM-MAS, the coverage matrix needs a reformulation. In traditional SBFL, the coverage matrix records whether a component is executed in a test case. Translated directly to LLM-MAS, each entry would indicate whether a role participates in a given execution. However, because most LLM-MAS frameworks invoke nearly all roles in every execution by design, such a binary matrix would yield identical entries across passing and failing runs, resulting in tied suspiciousness scores. To address this, we replace binary entries with weights that reflect how actively each role participates in a given execution, following prior work that resolves ties using frequency information [8].

RQ2: Spectrum Suitability. Of the three spectrum representations evaluated, Spectrum 3 proves the most informative. By filtering messages according to their cosine similarity with the final code artifact, it focuses on agent communications that are eventually reflected in the output. Spectra 1 and 2, by contrast, both rely on raw message counts and neither produces stable or informative rankings. Taken together, these findings suggest that richer spectrum representations, ones that capture the semantic content of agent communications rather than their volume alone, are more likely to yield accurate suspiciousness rankings.

RQ3: Fault Localization Accuracy. Our results indicate that SBFL can identify the most suspicious role with moderate reliability, but struggles to produce accurate rankings beyond the top position. Spectrum 3 achieves a Top-1 accuracy of 60%, correctly identifying the most suspicious role in three of the five tasks, while Spectra 1 and 2 reach at most 20% and 40% accuracy, respectively. Meanwhile, Top-3 accuracy is zero across nearly all spectra and configurations. Our results further show that the choice of SBFL formula has negligible impact on localization performance and incorporating additional execution runs during SBFL analysis does not consistently improve localization accuracy.

5 Discussion

5.1 Spectrum Design

The three spectrum representations introduced in this paper are designed to be LLM-MAS framework- and benchmark task- agnostic. To apply them, only two inputs are needed, namely an execution log in which each message is attributed to a named agent role and a binary outcome indicator. No assumption is made regarding the internal architecture of the LLM-MAS under study, the number of roles, or the nature of the benchmark task. SWE-bench is chosen primarily to avoid one additional labelling step, as pass/fail outcomes are determined automatically by the ground truth test suite [14]. The spectrum representations themselves can be applied directly to other LLM-MAS frameworks, provided that execution logs are role-attributed and run outcomes can be evaluated as either passing or failing.

5.2 Threats to Validity

Reliability of LLM-as-a-judge The most significant threat to validity is the use of an LLM-as-a-judge to establish ground truth faulty roles. Since no faults are injected and HyperAgent fails naturally, there is no simple, resource-efficient way to verify which role truly caused a failure. LLM judge's attributions are subject to well-known LLM limitations, including sensitivity to prompt phrasing [17], context length [12], and inconsistency across runs [18]. Zhang et al. [12] report that even the best automated failure attribution method achieves only around 53.5% agent-level accuracy. This suggests that a non-trivial portion of what we consider ground truth labels in our experiments may be incorrect, and a spectrum that correctly identifies a faulty role may still be penalized if the LLM judge attributes blame elsewhere.

Use of Closed-Source Models Both LLM-MAS execution and ground truth construction rely on DeepSeek-V4-Flash, a closed-source API model by DeepSeek. Additionally, Spectrum 3 as defined in 3.2 depends on OpenAI's text-embedding-3-small model for semantic similarity scoring. These dependencies limit experimental control and we acknowledge that open-source, locally-hosted alternatives would offer greater transparency and reproducibility. However, given the limited timeline, the setup effort, and the infrastructure requirements that locally hosting LLM would entail, closed-source models was a practical choice that allowed us to focus on the research questions rather than model setup.

External Validity and Generalizability Our evaluation is conducted on only five benchmark tasks and a single LLM-MAS framework, HyperAgent. While the spectrum definitions are designed to be framework- and task- agnostic, it is not possible to determine from these results alone whether the observed localization performance generalizes to other LLM-MAS and to tasks outside SWE-bench. This motivates a larger-scale evaluation for future work.

6 Conclusions and Future Work

This paper investigated whether SBFL can be adapted to identify faulty agent roles in LLM-MAS. Three spectrum representations were evaluated across five SWE-bench Verified tasks using Tarantula, Ochiai, and DStar2 as suspiciousness scoring functions. Among them, semantically filtering messages by alignment with the final code artifact prior to counting (Spectrum 3) achieved the strongest Top-1 localization accuracy of 60%, consistently outperforming raw counts in Spectra 1 and 2. However, no spectrum variant produced reliable rankings beyond the top position, no SBFL formula consistently outperformed others, and additional execution runs did not always improve accuracy. These results raise the question of whether SBFL, even with better spectrum representations, is fundamentally well-suited to LLM-MAS and motivate three directions for future work.

First, the evaluation should be scaled to more tasks and multiple LLM-MAS frameworks to assess whether the findings generalize beyond HyperAgent and SWE-bench. Second, improving ground truth quality should be prioritized. Replacing LLM-as-a-judge with fault injection or human-

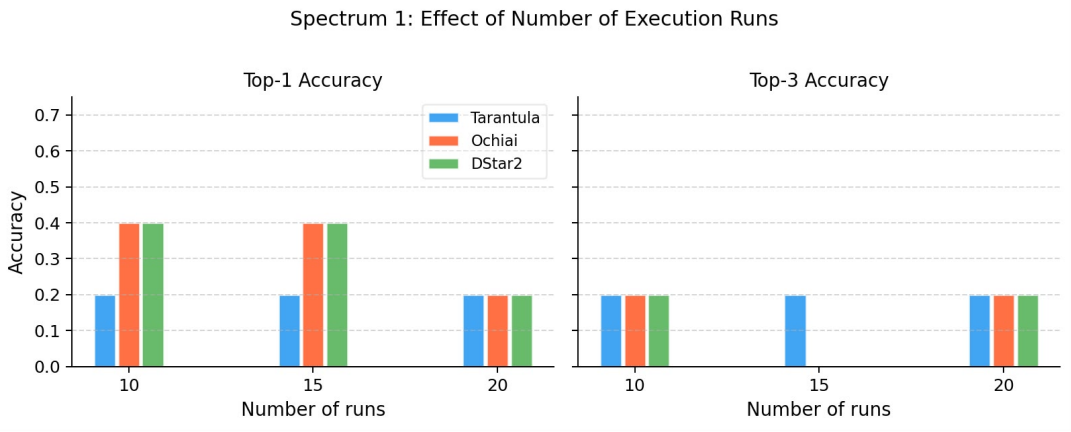

Figure 4: Spectrum 1 accuracy averaged over five tasks for varying number of execution runs and SBFL formula.

validated annotations would provide a more reliable evaluation, without which it is difficult to determine whether a spectrum is genuinely ineffective or is penalized by inaccurate judge labels. Third, richer spectrum representations that go beyond message volume and output overlap, such as capturing the inter-agent communication or tool invocation patterns, may be necessary to produce more accurate and actionable SBFL rankings.

7 Responsible Research

This research does not involve personal data or ethically sensitive information. All data used in our experiments is generated by running HyperAgent on a subset of SWE-bench Verified tasks, both of which are publicly available frameworks. Furthermore, to support reproducibility, all key experimental choices are discussed in Section 4.1. The source code and instructions to run the experiments are made available in a public repository [3]. We acknowledge that repeating our experiments may yield slightly different results, as LLM outputs are inherently non-deterministic.

Since manually annotating all execution traces would require substantial domain expertise and effort, it is impractical given the timeline and resource constraints of our research. As a result, we treat the outputs from the LLM-as-a-judge as ground truth labels and discuss the limitations and threats to validity this introduces in Section 5. Future studies should consider incorporating human-validated annotations, as label quality may affect the reliability of downstream evaluations, as discussed in Section 6.

Finally, we acknowledge the use of generative AI tools, namely Claude and ChatGPT, to help with improving the clarity and structure of this paper. All research ideas, methodological decisions, experimental designs, and interpretations are of our own, and we maintain responsibility for the accuracy of the presented content.

[3] https://github.com/nguyenlekhadan/role-based-SBFL

References

[1] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya G. Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025.

[2] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024.

[3] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024.

[4] Huy Nhat Phan, Phong X Nguyen, and Nghi DQ Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2406.11912*, 2024.

[5] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Softw. Eng.*, 42(8):707–740, 2016.

[6] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82:1780–1792, 11 2009.

[7] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques*, TAICPART-MUTATION '07, page 89–98. IEEE Computer Society, 2007.

[8] Hua Jie Lee, Lee Naish, and Kotagiri Ramamohanarao. Effective software bug localization using spectral frequency weighting function. In *2010 IEEE 34th Annual*

Computer Software and Applications Conference, pages 218–227, 2010.

[9] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge. 2024.

[10] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, page 273–282, New York, NY, USA, 2005. Association for Computing Machinery.

[11] W. Eric Wong, Vidroha Debroy, Yihao Li, and Ruizhi Gao. Software fault localization using dstar (d*). In *2012 IEEE Sixth International Conference on Software Security and Reliability*, pages 21–30, 2012.

[12] 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, 2025.

[13] Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Who is introducing the failure? automatically attributing failures of multi-agent systems via spectrum analysis, 2025.

[14] Carlos E. Jimenez, John Yang, et al. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[15] Tianyi Liu, Yi Li, Yiyu Zhang, Zhuangda Wang, Rongxin Wu, Xuandong Li, and Zhiqiang Zuo. Tailtracer: Continuous tail tracing for production use. 9(OOPSLA2), October 2025.

[16] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 807–817, New York, NY, USA, 2019. Association for Computing Machinery.

[17] Jingming Zhuo, Songyang Zhang, Xinyu Fang, Haodong Duan, Dahua Lin, and Kai Chen. Prosa: Assessing and understanding the prompt sensitivity of llms, 2024.

[18] Rajarshi Haldar and Julia Hockenmaier. Rating roulette: Self-inconsistency in llm-as-a-judge frameworks. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, page 24986–25004. Association for Computational Linguistics, 2025.