
RAFFLES: Reasoning-based Attribution of Faults for LLM Systems

Chenyang Zhu Spencer Hong Jingyu Wu Kushal Chawla Charlotte Tang
Youbing Yin Nathan Wolfe Erin Babinsky Daben Liu

Capital One

{chenyang.zhu, spencer.hong, jingyu.wu, kushal.chawla, yuhui.tang,
youbing.yin, nathan.wolfe, erin.babinsky, daben.liu}@capitalone.com

Abstract

We have reached a critical roadblock in the development and enhancement of long-horizon, multi-component LLM agentic systems: it is incredibly tricky to identify where these systems break down and why. Evaluation capabilities that currently exist today (e.g., single pass LLM-as-a-judge) are limited in that they often focus on individual metrics or capabilities, end-to-end outcomes, and are narrowly grounded on the preferences of humans. We argue that to match the agentic capabilities, evaluation frameworks must also be able to reason, probe, iterate, and understand the complex logic passing through these systems over long horizons. In this paper, we present RAFFLES - an evaluation architecture that incorporates reasoning and iterative refinement. Specifically, RAFFLES operates as an iterative, multi-component pipeline, using a central Judge to systematically investigate faults and a set of specialized Evaluators to assess not only the system's components but also the quality of the reasoning by the Judge itself, thereby building a history of hypotheses. We tested RAFFLES against several baselines on the Who&When dataset, a benchmark designed to diagnose the "who" (agent) and "when" (step) of a system's failure. RAFFLES outperforms these baselines, achieving an agent-step fault pair accuracy of over 43% on the Algorithmically-Generated dataset (a substantial increase from the previously published best of 16.6%) and over 20% on the Hand-Crafted dataset (surpassing the previously published best of 8.8%). These results demonstrate a key step towards introducing automated fault detection for autonomous systems over labor-intensive manual human review.

1 Introduction

As large language models (LLMs) evolve into complex, multi-component systems, a critical gap has emerged between what they can do and how we can effectively evaluate them. The rise of language-conditioned agentic architectures such as ReAct [1], Toolformer [2], and Reflexion [3] allows systems to plan, reason, and act over long time horizons. However, these new architectures also introduce novel failure modes that current evaluation strategies are not equipped to detect. Most current strategies are confined to isolated metrics that focus on overall outcomes and they struggle with longer contexts [4–6]. This leaves manual "detective work" as the only way to identify root cause errors. In long-horizon agentic systems, a single, subtle error can quickly cascade, creating a ripple effect of unexpected behaviors. Finding these root cause errors is a true "needle in the haystack" problem. To date, when such step-level evaluation is done manually, it takes on the order of many minutes to tens of minutes *per data instance* [4, 7, 8]. Such manual debugging is too costly and can become a bottleneck. Instead, automatic evaluations of multi-turn, multi-component systems must shift focus from end performance to better understanding *where* failures originate and *how* they form.

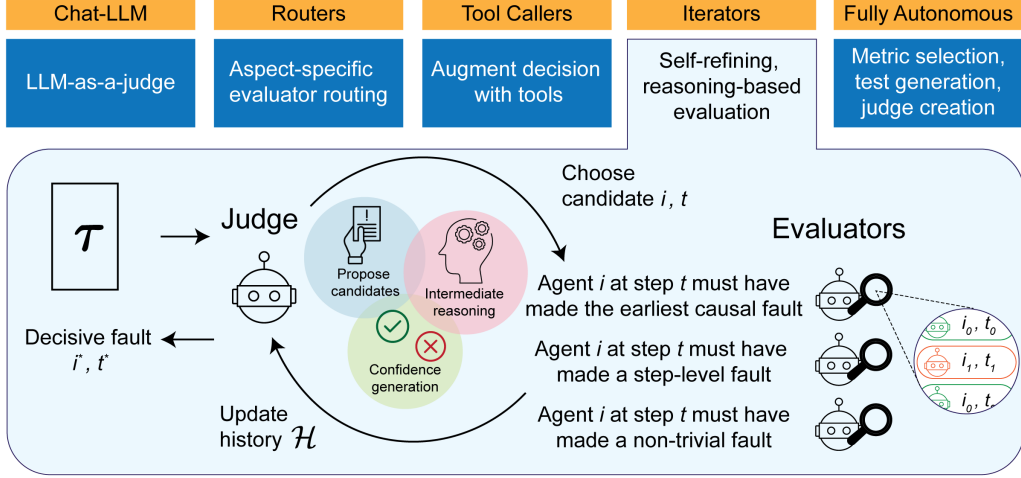


Figure 1: **Our proposed RAFFLES framework for multi-turn agentic evaluation by reasoning-based fault attribution.** Evaluating new LLM systems requires a new generation of evaluators that integrate multi-dimensional metrics, explainable judgments, and iterative refinement. RAFFLES leverages specialized Evaluators designed to assess candidate faults based on the criteria of a decisive fault. Each Evaluator takes in the full log τ and intermediate reasoning, which are passed to subsequent iterations until a decisive fault is determined. The presented evolution of LLM systems (orange) is inspired by a tutorial by Hassani, Karbasi, and Robey [13].

Automatic failure attribution within LLM multi-component agentic systems is an emerging area of focus for the evaluation community [7–10]. However, preliminary results have demonstrated limited success due to the challenges imposed by detecting faults in complex agentic systems. For example, the LLM-as-a-judge has become a cornerstone of automated evaluation due to its flexibility in approximating human evaluators [11, 12], and yet single-pass LLM evaluators struggle to detect faults within the long trajectories of agentic systems [7, 8]. Effective evaluation by which one can pinpoint faulty planning, logic, tool calls, calculations, code, etc. within multi-component systems *requires* its own set of capabilities (e.g., reasoning, planning, tool calling, iterating). At a recent ICML tutorial, a framework was introduced describing the progression from simple Chat-LLM, to Routers, Tool-Caller, Iterators, and, finally, Fully Autonomous systems [13]. We believe that evaluation systems must likewise evolve in a parallel manner: from single judgment Chat-LLMs, to aspect-specific evaluator routing, deciding with tools, iterators that loop and refine, and eventually, fully autonomous evaluators. We present this evolution of evaluators in Figure 1.

The need to evolve evaluation strategies to meet the demands of multi-component LLM systems can be demonstrated via a simple example of implementing Retrieval-Augmented Generation (RAG). RAG systems are conventionally judged by the final summary’s fidelity to an ideal summary [14, 15]. A more insightful evaluation, however, would be to determine the point of failure (e.g., retrieval vs. generation) and characterize its nature (e.g., irrelevant retrieval vs. incoherent retrieved sets), to provide a more accurate depiction of the system’s capabilities and shortcomings. Now, suppose we also extend this system to include different tools for retrieval and a reflection loop to self-correct those retrieved documents (i.e. [16]); every additional component that gets added increases the scope of agent activity, the interaction effects, and the multitude of decisions that need to be understood and evaluated.

In this paper, we attempt to realize this evolution in evaluation capabilities. We compare different classes of evaluation architectures (Figure 1) with the explicit goal of detecting decisive, trajectory-altering faults in multi-component systems. First, we extend and structure the reasoning capabilities of our LLM-as-a-judge by introducing Evaluators. Second, we introduce iterative loops by which our Judge can reflect on the confidence scores of the Evaluators and adjust its own assessments. We test these capabilities on the Who&When benchmark [7], which consists of agentic logs and agent-step fault pairs for a set of queries from GAIA [17] and AssistantBench [18]. Our experiments demonstrate

that structured iterative reasoning provides a significant and robust performance advantage over existing methods, consistently across diverse model families. Using Llama 3.3 70b model, RAFFLES improved the step-level accuracy on fault attribution from a previously published best of 16.6% to 43.6% on the Algorithmically-Generated dataset, and from a previously published best of 8.8% to 20.7% on the Hand-Crafted dataset.

2 Related Work

Reasoning and automated evaluators. The automated evaluation of LLMs (and agentic systems) has been enabled by (a) the extensive work done by researchers to introduce the LLM-as-a-judge [11, 12, 19, 20], and (b) recent advances in the reasoning capabilities of LLMs [21–24]. Enhancements of the LLM-as-a-judge include new methods that incorporate planning within the prompt [25], self-check reasoning [26], debate [27], and juries as judges [28]. Our contribution introduces a unique evaluation architecture that incorporates numerous reasoning advances; we introduce a Judge and a set of Evaluators that incorporate structured reasoning via problem decomposition [29, 30], natural language confidence estimation [31–33], as well as iterative refinement [3, 34].

Fault attribution in agentic systems. Numerous evaluation frameworks [7, 8, 35–38] have shown preliminary success evaluating agentic benchmark datasets like GAIA, SWEbench, and Assistant-Bench [17, 18, 39]. Frameworks that detect trajectory-breaking faults include (a) a multi-agent debugging tool [4], (b) a prompt-customized LLM-as-a-judge implementation [8], and (c) a comparison of methods including the LLM-as-a-judge as well as binary search and Step by Step implementations - the latter two allowing for multiple passes to isolate the step-level failure [7]. In our work, we specifically focus on step-level fault detection and explore the benefits of fault detection within a multi-step tolerance, given the interaction effects that occur between agent steps. Furthermore, we introduce iterative refinement as a means to better detect faults within long-context traces, especially for those that are more likely to get "lost in the middle" [5].

3 Background

Following Zhang *et al.* [7], we denote $M = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \varphi \rangle$ as a turn-based multi-agent system. Here, $\mathcal{S}$ is the set of possible states s . $\mathcal{A}$ is the global action space of action a . Each agent $i \in \mathcal{N}$ can perform actions from some subset $\mathcal{A}_i \subseteq \mathcal{A}$. $\varphi(t)$ is a function that indicates which agent is active at time t , ensuring that only one agent is active at time t .

A full trajectory τ can be written as $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$ where T is a terminal time step or when the system enters a terminating state. The success or failure of a trajectory is determined by a binary outcome function $Z(\tau)$:

$$Z(\tau) = \begin{cases} 1 & \text{if } \tau \text{ succeeds,} \\ 0 & \text{if } \tau \text{ results in failure.} \end{cases}$$

In practice, Z usually represents an external evaluator, such as an LLM-based judge, that evaluates the output of the pipeline τ , assessing whether the final outcome meets the desired objective [17, 40]. Given the dataset, we assume that there exists one or more τ^* such that $Z(\tau^*) = 1$. When a system fails ($Z(\tau) = 0$), we seek to understand where and when the failure occurs within the trajectory. This requires a causal analysis to determine which action, if altered, would have changed the outcome.

To this end, we establish a precise hierarchy of faults to progress from local, procedural errors to the single, causally significant event that led to the system’s failure.

Definition 3.1 (Step-Level Fault). *A step-level fault occurs at time t if the action a_t is deemed incorrect for a given state s_t by a local judge function f that takes in (a_t, s_t) . Specifically, f produces a probability $c \in [0, 1]$ representing the likelihood of error. A fault exists if $c = f(a_t, s_t) > \varepsilon$.*

The function f is usually a case-by-case LLM-judge that follows a certain taxonomy, most commonly hallucination or execution error (e.g. [8, 10]). However, a step-level error may not be causally responsible for the final failure. To establish causality, we must determine if a step-level fault had a tangible impact on the trajectory’s result. To this end, we define two mutually exclusive fault concept.

Definition 3.2 (Causal Fault). *Let τ be a trajectory with a failed trajectory ($Z(\tau) = 0$) containing a step-level fault a_t ($f(s_t, a_t) > \varepsilon$) such that an intervention that replaces a_t with $\tilde{a}_t$ results in a*

successful trajectory. Formally, a_t is a causal fault if

$$Z(\tau) = 0 \quad \wedge \quad \exists \tilde{a}_t \text{ s.t. } Z(\tau|do(a_t := \tilde{a}_t)) = 1$$

Here $\tau|do(a_t := \tilde{a}_t)$ denotes the counterfactual trajectory that unfolds after the intervention at time t , borrowing notation from causal inference literature [41]. This definition isolates actions that were causally responsible for the failure. Our definition of causal fault is similar to the definition of decisive fault in Who&When [7]. However, a trajectory may contain step-level faults that are not causally responsible for the final outcome, which we define as a trivial fault.

Definition 3.3 (Trivial Fault). Let τ be a trajectory with a failed trajectory ($Z(\tau) = 0$) containing a step-level fault a_t ($f(s_t, a_t) > \varepsilon$) such that no intervention replacing a_t with $\tilde{a}_t$ can result in a successful trajectory. Formally, a_t is a trivial fault if

$$Z(\tau) = 0 \quad \wedge \quad \forall \tilde{a}_t \text{ s.t. } Z(\tau|do(a_t := \tilde{a}_t)) = 0$$

This occurs if a fault was minor, corrected later, or if a subsequent, unavoidable fault later in the trajectory would have caused a failure regardless.

Finally, we define the decisive fault as the first causal fault in the trajectory. It is the earliest point at which an intervention would have salvaged the outcome, making it the primary origin of the failure. In the annotation guide in Who&When benchmark [7], annotators were asked to first think of the most serious mistaken agent, implicitly considering the fault’s severity. However, prior definitions of fault attribution have not formally accounted for this concept. To address this gap, we extend the definition of fault attribution by incorporating a step-level function, f , to quantify the severity of a fault.

Definition 3.4 (Decisive Fault Attribution). The decisive fault is the first causal fault to occur in a trajectory. The process of identifying it can be formulated as a constrained optimization problem:

$$\begin{aligned} t^* &= \arg \min_t t \\ \text{subject to } & (a) \ f(s_t, a_t) > \varepsilon \\ & (b) \ Z(\tau|do(a_t := \tilde{a}_t)) = 1, \text{ for some } \tilde{a}_t \in \mathcal{A} \end{aligned}$$

Definition 3.4 refines the theoretical construct of a decisive fault from Who&When by stipulating that it must be a non-trivial, high-severity error. This revised definition not only aligns more closely with the annotation guidelines for Who&When but also motivates the design of our RAFFLES framework.

4 RAFFLES

Given a system trajectory τ , we designed RAFFLES (Algorithm 1) as a Judge-Evaluator iterative LLM system to attribute fault based on Definition 3.4, which we distill into 3 necessary criteria.

1. **Primacy.** The action of agent i at step t must be the earliest causal fault, satisfying the goal of finding the min of t .
2. **Fault Condition.** The action of agent i at step t must constitute a step-level fault, satisfying constraint (a).
3. **Causality.** The agent i ’s mistake must be a causal fault and not a trivial fault, satisfying constraint (b).

The design of the Judge and Evaluators is specifically tailored to facilitate structured reasoning of these three criteria with confidence.

Judge: Judge J receives relevant execution logs τ and proposes the most likely candidate for the decisive fault, consisting of the agent-step pair (i, t) and 3 criteria-driven rationales $R_j = \{r_j^1, r_j^2, r_j^3\}$ for its candidate selection. To enable structured reasoning, Judge provides rationales separately for the three criteria of decisive faults. Judge is also asked to reference the log as needed for clarity, so as to improve the faithfulness of the rationale. See details of the Judge prompt in Appendix F.3.

Evaluator: Each Evaluator E_p , for $p \in \{1, 2, 3\}$ evaluates one of the three criteria for decisive faults given the proposed agent-step pair (i, t) , τ , and the rationale r_j^p from the Judge. Each E_p returns a

rationale r_e^p assessing the criterion-specific soundness of the Judge’s rationale r_j^p and a confidence score c_e^p between 0 and 100. We also designed an additional rule-based Evaluator $p = 4$ to validate whether the proposed agent-step (i, t) pair is consistent with the log τ . The sum of confidences are denoted as $C = \sum_p c_e^p$ and the rationales are denoted as $R_e = \{r_e^p; p = \{1, 2, 3, 4\}\}$. See details of the Evaluator prompt in Appendix F.3.

To facilitate the iterative reasoning, the output of the Evaluators is appended to a memory component $\mathcal{H}$. $\mathcal{H}$ is then fed back to the Judge in the subsequent iteration, enabling it to refine its candidate agent-step pair (i, t) selection strategy.

The iteration concludes (TERMINATE) and the proposed agent-step pair (i, t) is considered the decisive fault given the log τ and the evidence in $\mathcal{H}$ when either (a) C is greater than a threshold of 350, or (b) when a predefined maximum number of iterations K is reached and we take the candidate with the highest confidence in $\mathcal{H}$. We set $K = 2$ following the results of an ablation study in Figure 2b. An illustrative example of the iterative process can be found in Appendix A.

Overall, RAFFLES relies on the iterative and explicit reasoning process between the Judge and the Evaluators, tailored to solve the constraint optimization problem defined in 3.4. Given the challenges imposed by long-horizon, multi-component systems, we equip RAFFLES with more capabilities to increase overall evaluation accuracy. We introduce a Judge that uses structured reasoning to predict decisive faults, while a set of Evaluators check those predictions against unique criteria and report their agreement using natural language confidence estimates. The Judge then incorporates this feedback to iteratively refine decisive fault predictions to enable more accurate and reliable fault detection.

Algorithm 1 RAFFLES: Iterative Reasoning-based Fault Attribution

Require: Trajectory τ , max iterations K

```

1:  $\mathcal{H} \leftarrow \emptyset$  {Initiate memory component for history}
2:  $k \leftarrow 0$ 
3: while not TERMINATE( $\mathcal{H}$ ) and  $k < K$  do
4:    $(i, t), R_j \leftarrow \text{Judge}(\tau, \mathcal{H})$  {Propose agent-step pair and rationale from  $\tau$ }
5:    $C, R_e \leftarrow \text{Evaluator}(i, t, R_j, \tau)$  {Invoke Evaluator  $E_p$ }
6:    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(C, R_e)\}$  {Update history}
7:    $k \leftarrow k + 1$ 
8: end while
9: if TERMINATE( $\mathcal{H}$ ) then
10:  return  $(i^*, t^*)$ , the decisive fault identified from  $\mathcal{H}$ 
11: else
12:  return None {No decisive fault found}
13: end if
```

5 Experiments

5.1 Baseline methods

We perform comprehensive experiments on RAFFLES with Llama-3.3-70B-Instruct¹, Llama-3.1-8B-Instruct², Mixtral-8x22B-Instruct-v0.1³, and gpt-oss-20b model⁴. All of the models are deployed on NVIDIA A100 nodes using vLLM for inference optimization without quantization. For all experiments except Mixtral, a greedy search is used with a maximum context length of 128k tokens; Mixtral uses a maximum context length of 64k tokens. We include the GPT-4o results from Zhang *et al.* [7] in our experiment result table as a comparison. For gpt-oss-20b, we set reasoning mode as *low*. We empirically find that other modes of reasoning creates long, incoherent outputs, leading to context limit errors. This finding aligns with the non-scaling trends documented in recent literature [42], and we are now actively investigating this phenomenon in the context of long-horizon evaluations.

¹<https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>

²<https://huggingface.co/meta-llama/Llama-3.1-8B>

³<https://huggingface.co/mistralai/Mixtral-8x22B-Instruct-v0.1>

⁴<https://huggingface.co/openai/gpt-oss-20b>

In addition to RAFFLES, we implement four baselines categorized by the evaluation system hierarchy shown in Figure 1, covering prior work and a strong Tool-Caller baseline proposed by us. The first is a one-shot **LLM-as-a-judge** from [7], which directly evaluates the entire log and problem statement to find the faulty step. The second class of evaluators is LLM routers proposed in Who&When [7], including a **Step by Step** method that examines the log sequentially, stopping when an error is detected and a **binary search** method that iteratively determines if an error lies in the upper or lower half of the log to narrow the search. Finally, we propose a strong baseline **Tool-Caller** that uses a planner to decide which step to investigate, calling an LLM judge with the log index (e.g., `<tool>judge(id=1)</tool>`) up to three times. The prompts used for these baselines are available in Appendix F.

5.2 Dataset

Our experiments are conducted on the Who&When dataset [7], a benchmark specifically created for fault attribution in multi-agent systems. The dataset is composed of two distinct subsets: (a) an Algorithmically-Generated dataset, which has 191 unique agents across 126 logs, and (b) a Hand-Crafted dataset with 5 unique agents across 58 logs. On average, Algorithmically-Generated task logs have 8.6 steps while Hand-Crafted logs have 50 steps (see Appendix B Table 4). The Hand-Crafted dataset was specifically designed to be more challenging and reflect realistic system failures. Each log in the dataset includes the initial problem statement, a complete trace of agent-to-agent communications, the ground truth answer of the initial problem, and an annotated label pinpointing the erroneous agent and the specific step number of the failure. All of our experimental results attribute faults without any access to the ground-truth information.

5.3 Evaluation Metrics

We propose two metrics for fault attribution. (1) **Strict Step-Level Accuracy** (or simply Step-Level Accuracy): This is the primary metric, the proportion of test cases where the system correctly predicts the exact step number t of the fault within the trajectory τ . This serves as our most stringent measure of performance, requiring precise identification of the failure point. (2) **Tolerant Step-Level Accuracy**: This secondary metric measures accuracy within a tolerance window of k steps. A prediction $\hat{t}$ is considered correct if it falls within k steps of the ground-truth fault step (i.e. $|\hat{t} - t_{gt}| \leq k$). This metric, known as step-level accuracy with tolerance $\pm k$ steps, is a good indicator of practical utility, as pinpointing a fault to a small window (e.g., $k \leq 2$) is often sufficient to guide efficient manual inspection and debugging of interacting components.

Prior work has also reported an agent-level accuracy metric [7]. However, in our analysis, we found significant label imbalance at the agent-level: 97% of all agent events are the WebSurfer and the Orchestrator, and, in 57% of all cases, the WebSurfer agent is the ground truth cause of failure. Thus, a trivial baseline could achieve an agent-level accuracy of 57%, making agent-level accuracy less meaningful. Hence, our experiments primarily focus on step-level metrics described above. We discuss these findings in more detail in Appendix B. We present agent-level comparisons in Appendix C for completeness.

6 Results

Tables 1 and 2 detail the step-level accuracy on the Algorithmically-Generated and Hand-Crafted datasets, respectively, using a maximum of $K = 2$ iterations. To validate our setup, we confirmed that our baseline performance is comparable to the original results published in Who&When [7], except for the Step by Step method, for which we consistently got 0 step level accuracy on the Hand-Crafted dataset. We included results from the original Who&When paper in the **Zhang et al.** [7] column for reference.

RAFFLES detects decisive faults better than existing methods across diverse model families. On the Algorithmically-Generated dataset, RAFFLES outperforms our strongest baseline, Tool-Caller, by 31% with Llama 3.3 70B and 51% with gpt-oss-20b. The performance superiority is further validated on the more challenging Hand-Crafted dataset, where RAFFLES surpasses Tool-Caller by 53% (Llama 3.3 70B) and 30% (gpt-oss-20b). These results collectively show that our framework’s iterative approach provides a universal performance lift across all benchmark models, from large-scale

Table 1: Step-level accuracy on Who&When Algorithmically-Generated dataset

Method	Llama 3.1 8B	Llama 3.3 70B	Mixtral-8x22b	gpt-oss-20b	Zhang <i>et al.</i> [7]
Chat-LLM	15.08	19.05	<u>20.63</u>	14.29	13.53
Routers					
Step by Step	5.56	6.35	10.32	11.90	15.31
Binary Search	0.79	4.76	0.79	23.81	16.59
Tool-Caller	<u>19.05</u>	<u>33.33</u>	17.46	<u>29.37</u>	-
RAFFLES	34.92	43.65	29.37	44.44	-

Table 2: Step-level accuracy on Who&When Hand-Crafted dataset

Method	Llama 3.1 8B	Llama 3.3 70B	Mixtral-8x22b	gpt-oss-20b	Zhang <i>et al.</i> [7]
Chat-LLM	6.90	6.90	1.75	3.45	3.51
Routers					
Step by Step	0.00	0.00	0.00	0.00	8.77
Binary Search	5.17	10.34	<u>10.34</u>	13.79	6.90
Tool-Caller	<u>18.97</u>	<u>13.56</u>	7.02	<u>17.24</u>	-
RAFFLES	25.86	20.69	15.79	22.41	-

Mixture-of-Experts (MoE) models with pre-trained reasoning capabilities to smaller models with limited pre-trained reasoning. Furthermore, as shown in Table 3, RAFFLES consistently outperforms our Chat-LLM baseline across nearly all tolerance thresholds in both datasets.

Methods using only partial trajectories as input context underperform. We find that methods with access to the end-to-end trajectories are more successful in fault attribution. The Step by Step approach, which only retains context of prior steps, struggles significantly on longer trajectories like those in the Hand-Crafted dataset where it consistently underperforms compared to the Chat-LLM baseline across nearly all model families. Similarly, the binary search method is moderately effective at localizing faults within long contexts but proves ineffective on shorter trajectories where its narrowing function may be unnecessary and negatively impactful. These results not only underscore the importance of maintaining a global, end-to-end view of the trajectory, but also the flexibility for focused, localized analysis. RAFFLES outperforms these baselines by utilizing the design where the Judge operates on the global context to form hypotheses and the Evaluators concentrate the reasoning process on the hypothesized agent-step fault candidate.

Performance degrades for longer trajectories. While a global context is necessary for robust fault attribution, our analysis reveals a central dichotomy: the effectiveness of these methods is negatively impacted by increasing trajectory length. As our experiments in Figure 2a reveal, a clear pattern emerges where step-level fault detection accuracy decreases as trajectories increase in context token length. This observation is directly attributable to the inherent challenge models face in accurately pinpointing a decisive fault within long, complex trajectories. However, the results in Figure 2a demonstrate that RAFFLES consistently outperforms the Chat-LLM baseline, a performance advantage that holds even for longer contexts.

Structured reasoning outperforms flexible tool calling. While the Tool-Caller baseline introduces an improvement over the Router and Chat-LLM baselines, it is still consistently outperformed by RAFFLES. We attribute this performance gap to the lack of procedural reliability and robust reasoning within the Tool-Caller’s Planner. Although designed for flexible and efficient decision-making, the Planner frequently fails to generate high-quality candidates for the Judge component to evaluate. A key advantage of RAFFLES is structured reasoning. As shown in Figure 2b, even with a single iteration $K = 1$, which constitutes a single-pass with a structured LLM judge, RAFFLES achieves a step-level accuracy of 27.59% on Hand-Crafted and 42.85% on Algorithmically-Generated dataset, outperforming all of our baselines using Llama 3.3 70b. This demonstrates that structured reasoning itself is inherently more effective than the flexible tool-calling implementation.

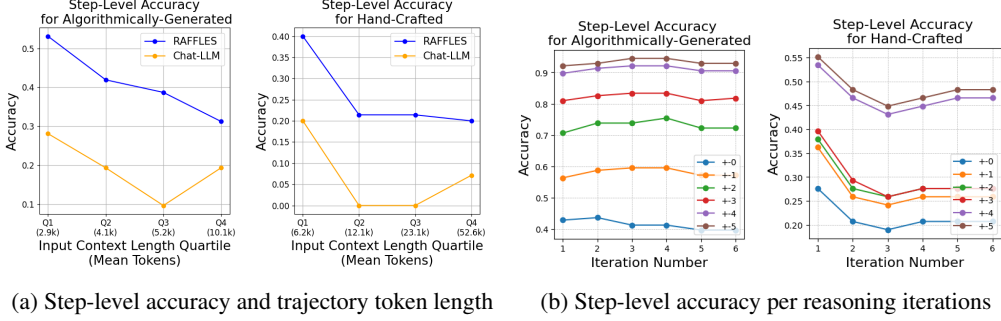


Figure 2: (a) Performance degrades with increased trajectory length. Quartile of token length is with respect to the corresponding dataset. (b) The effect of reasoning iteration on tolerant step-level accuracy ± 5 . All figures are using Llama 3.3 70b model.

Table 3: Tolerant step-level accuracy with $\pm t$ steps error tolerance using Llama 3.3 70b model

Dataset	Method	Acc.	± 1	± 2	± 3	± 4	± 5
Algorithmically-Generated	Chat-LLM (One Shot)	19.05	49.21	64.49	75.40	86.51	90.48
	RAFFLES (Iterator)	43.65	58.73	73.81	82.54	91.27	92.86
Hand-Crafted	Chat-LLM (One Shot)	6.90	12.07	20.69	31.03	32.76	44.83
	RAFFLES (Iterator)	20.69	25.86	27.59	29.31	46.55	48.28

Iterative improvement is not monotonic, necessitating an early stopping mechanism. Our results show that iterative reasoning does not monotonically improve accuracy, a finding consistent with Self-Refine [34], where they find performance improve for up to 3 iterations, suggesting diminishing returns with the number of iterations. We observe a similar, complex pattern in our experiments, as illustrated for the Llama 3.3 70B model in Figure 2b. On the Algorithmically-Generated dataset, the accuracy steadily improves from 70.35% to 75.40% with ± 2 tolerance by the fourth iteration. However, on the more challenging Hand-Crafted dataset, accuracy initially decreases before recovering and stabilizing at an equilibrium. This is likely caused by the Hand-Crafted dataset’s larger and more complex search space, which impedes convergence, as we detail in Appendix D. These findings collectively highlight the necessity of an early stopping mechanism, such as a maximum iteration count K , to cap the reasoning process.

7 Conclusion

The increasing complexity of agentic systems has revealed a significant methodological gap, where existing evaluation frameworks are insufficient for diagnosing system failures. Current approaches, like single-pass LLM-as-a-judge, often fall short because they focus on end-to-end outcomes, making them unable to pinpoint the origin of errors within complex trajectories. This exposes a central challenge for accurate, step-level fault attribution: identifying decisive faults requires the full context of a multi-component LLM system’s trajectory, yet LLMs struggle to reason effectively over long contexts.

To address this, we introduced RAFFLES, a novel iterative evaluation architecture designed to systematically identify these decisive faults. Our Judge-Evaluator iterative architecture builds a history of hypotheses and iteratively refines its reasoning, not only evaluating a candidate fault but also critiquing the reasoning behind its selection. By allowing error correction and iterative structured reasoning, which are central for problems that cannot be resolved in a single step, our results show a step-level accuracy of over 43% on the Algorithmically-Generated dataset (a substantial increase from the previously published best of 16.6%) and over 20% on the Hand-Crafted dataset (surpassing the previously published best of 8.8%). These results lays the foundation towards automated fault detection for LLM systems over time-consuming manual human review.

References

- [1] Shunyu Yao et al. “React: Synergizing reasoning and acting in language models”. In: *International Conference on Learning Representations (ICLR)*. 2023.
- [2] Timo Schick et al. “Toolformer: Language models can teach themselves to use tools”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 68539–68551.
- [3] Noah Shinn et al. “Reflexion: Language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 8634–8652.
- [4] Will Epperson et al. “Interactive debugging and steering of multi-agent ai systems”. In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 2025, pp. 1–15.
- [5] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *arXiv preprint arXiv:2307.03172* (2023).
- [6] Austin Xu et al. “Does context matter? contextualjudgebench for evaluating llm-based judges in contextual settings”. In: *arXiv preprint arXiv:2503.15620* (2025).
- [7] Shaokun Zhang et al. “Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems”. In: *arXiv preprint arXiv:2505.00212* (2025).
- [8] Darshan Deshpande et al. “TRAIL: Trace Reasoning and Agentic Issue Localization”. In: *arXiv preprint arXiv:2505.08638* (2025).
- [9] Jingwen Zhou et al. “SHIELDA: Structured Handling of Exceptions in LLM-Driven Agentic Workflows”. In: *arXiv preprint arXiv:2508.07935* (2025).
- [10] Mert Cemri et al. “Why do multi-agent llm systems fail?” In: *arXiv preprint arXiv:2503.13657* (2025).
- [11] Lianmin Zheng et al. “Judging llm-as-a-judge with mt-bench and chatbot arena”. In: *Advances in neural information processing systems* 36 (2023), pp. 46595–46623.
- [12] Jinlan Fu et al. “Gptscore: Evaluate as you desire”. In: *arXiv preprint arXiv:2302.04166* (2023).
- [13] Hamed Hassani, Amin Karbasi, and Alex Robey. *Jailbreaking LLMs and Agentic Systems: Attacks, Defenses, and Evaluations*. Tutorial presented at ICML, Vancouver, CA. 2025.
- [14] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries”. In: *Text summarization branches out*. 2004, pp. 74–81.
- [15] Kishore Papineni et al. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.
- [16] Zhenrui Yue et al. “Inference scaling for long-context retrieval augmented generation”. In: *arXiv preprint arXiv:2410.04343* (2024).
- [17] Grégoire Mialon et al. “Gaia: a benchmark for general ai assistants”. In: *The Twelfth International Conference on Learning Representations*. 2023.
- [18] Ori Yoran et al. “AssistantBench: Can Web Agents Solve Realistic and Time-Consuming Tasks?” In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 2024, pp. 8938–8968.
- [19] Xuechen Li et al. *AlpacaEval: An automatic evaluator of instruction-following models*. 2023.
- [20] Yang Liu et al. “G-eval: NLG evaluation using gpt-4 with better human alignment”. In: *arXiv preprint arXiv:2303.16634* (2023).
- [21] Hunter Lightman et al. “Let’s verify step by step”. In: *The Twelfth International Conference on Learning Representations*. 2023.
- [22] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.
- [23] Xuezhi Wang et al. “Self-consistency improves chain of thought reasoning in language models”. In: *arXiv preprint arXiv:2203.11171* (2022).
- [24] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in neural information processing systems* 36 (2023), pp. 11809–11822.
- [25] Swarnadeep Saha et al. “Learning to plan & reason for evaluation with thinking-llm-as-a-judge”. In: *arXiv preprint arXiv:2501.18099* (2025).
- [26] Ning Miao, Yee Whye Teh, and Tom Rainforth. “Selfcheck: Using llms to zero-shot check their own step-by-step reasoning”. In: *arXiv preprint arXiv:2308.00436* (2023).

- [27] Chi-Min Chan et al. “Chateval: Towards better llm-based evaluators through multi-agent debate”. In: *arXiv preprint arXiv:2308.07201* (2023).
- [28] Pat Verga et al. “Replacing judges with juries: Evaluating llm generations with a panel of diverse models”. In: *arXiv preprint arXiv:2404.18796* (2024).
- [29] Denny Zhou et al. “Least-to-most prompting enables complex reasoning in large language models”. In: *arXiv preprint arXiv:2205.10625* (2022).
- [30] Tushar Khot et al. “Decomposed prompting: A modular approach for solving complex tasks”. In: *arXiv preprint arXiv:2210.02406* (2022).
- [31] Stephanie Lin, Jacob Hilton, and Owain Evans. “Teaching models to express their uncertainty in words”. In: *arXiv preprint arXiv:2205.14334* (2022).
- [32] Miao Xiong et al. “Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms”. In: *arXiv preprint arXiv:2306.13063* (2023).
- [33] Katherine Tian et al. “Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback”. In: *arXiv preprint arXiv:2305.14975* (2023).
- [34] Aman Madaan et al. “Self-refine: Iterative refinement with self-feedback”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 46534–46594.
- [35] Mingchen Zhuge et al. “Agent-as-a-judge: Evaluate agents with agents”. In: *arXiv preprint arXiv:2410.10934* (2024).
- [36] Ma Chang et al. “Agentboard: An analytical evaluation board of multi-turn llm agents”. In: *Advances in neural information processing systems* 37 (2024), pp. 74325–74362.
- [37] Zhiwei Liu et al. “MCPEval: Automatic MCP-based Deep Evaluation for AI Agent Models”. In: *arXiv preprint arXiv:2507.12806* (2025).
- [38] Jiayi Pan et al. “Autonomous evaluation and refinement of digital agents”. In: *arXiv preprint arXiv:2404.06474* (2024).
- [39] Carlos E Jimenez et al. “Swe-bench: Can language models resolve real-world github issues?”. In: *arXiv preprint arXiv:2310.06770* (2023).
- [40] Jian Xie et al. “Travelplanner: A benchmark for real-world planning with language agents”. In: *arXiv preprint arXiv:2402.01622* (2024).
- [41] Judea Pearl. “Causal diagrams for empirical research”. In: *Biometrika* 82.4 (1995), pp. 669–688.
- [42] Sohee Yang et al. “How Well Can Reasoning Models Identify and Recover from Unhelpful Thoughts?”. In: *arXiv preprint arXiv:2506.10979* (2025).

Appendix

A Example of RAFFLES reasoning process

An example RAFFLES reasoning process can be found in the Figure below.

Question: 'How many High Energy Physics - Lattice articles listed in January 2020 on Arxiv had ps versions available?'

An agentic system's execution log to answer the question

Data_Extraction_Expert

Plan for solving the task\n1. Extract the list of High Energy Physics - Lattice articles from Arxiv for January 2020.\n2. Analyze the extracted data to identify which articles have ps versions available. \n3. Count the number of articles with ps versions available.

Verification_Expert

```
```python
from functions import arxiv_search
Step 1: Extract the list of High Energy Physics - Lattice articles from Arxiv for January 2020
query = "cat:hep-lat AND submittedDate:[2020-01-01 TO 2020-01-31]"
max_results = 1000

to cover all possible entries
articles = arxiv_search(query, max_results, sortby='submittedDate')

Step 2: Analyze the extracted data to identify which articles have ps versions available
ps_count = sum(1 for article in articles if 'ps' in article.get('entry_id', ''))

Output the result
print(ps_count)
```
```

Verification expert made a mistake by assuming ps version must exist in entry_id

Computer_terminal

exitcode: 0 (execution succeeded)\nCode output: 0\n

Verification_Expert:

Therefore, after careful verification, I conclude that the number of High Energy Physics - Lattice articles listed in January 2020 on Arxiv that had ps versions available is indeed 0.

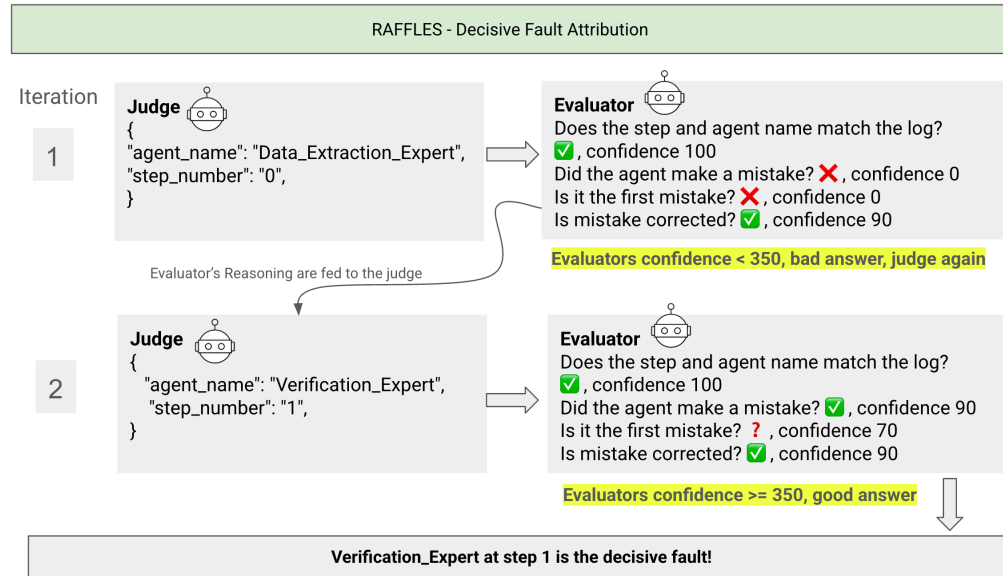


Figure 3: Example log from Who&When dataset, and how RAFFLES iterative reasoning process achieves the correct decisive fault.

Question: 'How many High Energy Physics - Lattice articles listed in January 2020 on Arxiv had ps versions available?'

Iteration 1

Judge

```
{
  "agent_name": "Data_Extraction_Expert",
  "step_number": "0",

  "mistake_reason": "The Data_Extraction_Expert made a mistake by stating that no High Energy Physics - Lattice articles were found for January 2020 on Arxiv, which led to the incorrect conclusion that none of these articles had ps versions available...",

  "first_mistake": "This is the first mistake step that relates to the final wrong outcome because it sets the premise for the rest of the conversation, where the other agents build upon this incorrect assumption...",

  "mistake_not_corrected": "The mistake was not corrected by or correctable by later agents because the subsequent steps relied on the initial incorrect data extraction. ... Therefore, the mistake was perpetuated throughout the conversation without correction."
}
```

Evaluators

```
Does the step and agent name match the log?
{
  "reason": "Agent name and step number match the task log's content.",
  "confidence": "100"

Did the agent make a mistake?
{
  "reason": "The log shows that the Data_Extraction_Expert provided a general task description and a plan for solving the task, but did not actually extract any data or make a claim about the number of articles found. The actual extraction and analysis were performed by the Verification_Expert at step 1, who wrote and executed the Python code to search for articles and count those with ps versions available.",
  "confidence": "0"

Is it the first mistake?
{
  "reason": "The actual execution and potential mistakes occur in later steps, particularly in the interaction with the Computer_terminal and the analysis performed by the Verification_Expert.",
  "confidence": "0"

Is mistake corrected?
{
  "reason": "The argument provided by the partner suggests that the mistake was not corrected by or correctable by later agents because ...",
  "confidence": "90"
}
```

Evaluators confidence < 350, bad answer

Iteration 2

Judge modifies the mistaken agent, and the reasoning based on the checker's rationale

```
{
  "agent_name": "Verification_Expert",
  "step_number": "1",

  "mistake_reason": "The Verification_Expert made a mistake by incorrectly assuming that the absence of 'ps' in the article's entry_id indicates the absence of a ps version... ",

  "first_mistake": "This is the first mistake step that relates to the final wrong outcome because it is the point where the actual data extraction and analysis occurred...",

  "mistake_not_corrected": "The mistake was not corrected by or correctable by later agents because the subsequent steps, including the execution by the Computer_terminal and the final verification, relied on the output of the Verification_Expert's analysis ..."
}
```

```
Does the step and agent name match the log?
{
  "reason": "Agent name and step number match the task log's content.",
  "confidence": "100"

Did the agent make a mistake?
{
  "reason": "The argument provided by the partner is sound because it correctly identifies a potential flaw in the Verification_Expert's assumption ...",
  "confidence": "90"

Is it the first mistake?
{
  "reason": "The mistake, if any, might have occurred earlier, such as in the formation of the query or in the interpretation of the task requirements. The partner's argument does not provide sufficient evidence to support the claim that step 1 was the first mistake, nor does it consider alternative explanations for the outcome.",
  "confidence": "70"

Is mistake corrected?
{
  "reason": "The argument provided by the partner for 'how this mistake was never corrected afterwards' is sound because ...",
  "confidence": "90"
}
```

Evaluators confidence >= 350, good answer

Verification_Expert at step 1 is the decisive fault

Figure 4: Continued example of RAFFLES’s iterative reasoning process between Judge and Evaluator. Each reasoning block from the Judge, showcased in different colors, will be sent to different Evaluators specifically. We instruct the Evaluators to not only focus on whether the reasoning given by the Judge is sound, and give a confidence score of the soundness of each rationale.

B Discussion on Who&When dataset

The Who&When dataset is one of the first open-source datasets to address the issue of fault attribution. It advances the field by giving a new perspective on evaluating language-conditioned agentic systems.

Table 4: Basic statistics of the Who&When dataset.

| Statistic | Who&When | |
|---------------------|---------------------------|--------------|
| | Algorithmically-Generated | Hand-Crafted |
| Number of Samples | 126 | 58 |
| Avg. Steps | 8.72 | 50.60 |
| Avg. Log Token Size | 3.24K | 18.68K |

Table 5: Agents with the most frequent faults in the Who&When dataset.

| Algorithmically-Generated | | Hand-Crafted | |
|---------------------------|-------|---------------|-------|
| Mistake Agent | Count | Mistake Agent | Count |
| Verification_Expert | 18 | WebSurfer | 33 |
| PythonDebugging_Expert | 7 | Orchestrator | 18 |
| DataAnalysis_Expert | 6 | Assistant | 4 |
| Validation_Expert | 5 | FileSurfer | 3 |
| DataVerification_Expert | 5 | | |

Table 6: Top-5 steps where the most mistakes occur in the Who&When dataset

| Algorithmically-Generated | | Hand-Crafted | |
|---------------------------|-------|--------------|-------|
| Step | Count | Step | Count |
| 1 | 34 | 8 | 9 |
| 0 | 20 | 4 | 9 |
| 5 | 14 | 12 | 8 |
| 3 | 13 | 9 | 3 |
| 8 | 12 | 16 | 3 |

We show some high-level statistics for this dataset in Table 4, as well as the most frequent step-level and faults in Table 5 and Table 6, respectively.

While the Hand-Crafted dataset has more steps on average (55 steps when compared to the Algorithmically-Generated average of 8 steps), there are only 5 possible agents across all steps and logs. Specifically, the WebSurfer agent is the decisive fault in 33 out of 58 instances. A trivial system that always guesses the WebSurfer can achieve at least 57% accuracy. Another factor impacting agent-level detection in the Hand-Crafted dataset is that 97% of the agent events are the WebSurfer and the Orchestrator. Epperson et al [4] found that there’s some nuance to detecting decisive faults at an agent-level, given that on some occasions, they could only correct a faulty WebSurfer instance by fixing the *preceding* planning step by the Orchestrator. This interactive nature between agents indicates that there is some subjectivity in assigning faults between planning and executing agents, and therefore, there may be some benefit to detecting agent- and step-level faults at some level of coarseness, or tolerance, rather than precisely at one step.

In addition, we’ve also found that 3 cases out of 126 in the Algorithmically-Generated dataset and 3 cases out of 58 in the Hand-Crafted dataset have inconsistencies where the agent-step pair in the ground truth does not correspond to the actual agent-step pair in the log. Either the agent name in the ground truth is erroneous, or the step number is incorrect. The specific instances are 59.json, 15.json, 14.json in Algorithmically-Generated dataset, and 22.json, 20.json, 49.json in Hand-Crafted dataset. Fixing these erroneous cases is outside of the scope of this paper, but we advise future researchers to take notice that these erroneous cases exist.

C Full Results on Who&When Dataset

In this section, we show both agent-level and step-level results on Who&When dataset with different model families of Llama 3.3 70b (Table 7), Llama 3.1 8b (Table 8), Mixtral-8x22b (Table 9) and GPT-oss-20b (Table 10).

RAFFLES consistently ranks first or second across model families on both the Algorithmically-Generated and Hand-Crafted datasets for both step-level and agent-level detection.

Table 7: Results of Llama 3.3 70b model on Who&When dataset

| Method | Algorithmically-Generated (126) | | Hand-Crafted (58) | |
|----------------------------|---------------------------------|--------------|-------------------|----------------------|
| | Step-level | Agent-level | Step-level | Agent-level |
| Chat-LLM (One Shot) | 19.05 | <u>55.56</u> | 6.90 | 63.79 (37/58) |
| Routers | | | | |
| Step by Step | 6.35 | 23.02 | 0.00 | 13.56 |
| Binary Search | 4.76 | 32.54 | 10.34 | 53.45 |
| Tool-Caller | <u>33.33</u> | 46.03 | <u>13.56</u> | 50.85 |
| RAFFLES (Iterators) | 43.65 | 61.90 | 20.69 | <u>60.34</u> (35/58) |

Table 8: Results of Llama 3.1 8b model on Who&When dataset

| Method | Algorithmically-Generated (126) | | Hand-Crafted (58) | |
|----------------------------|---------------------------------|-----------------------|-------------------|----------------------|
| | Step-level | Agent-level | Step-level | Agent-level |
| Chat-LLM (One Shot) | 15.08 | 53.97 (68/126) | 6.90 | 62.07 (36/58) |
| Routers | | | | |
| Step by Step | 5.56 | 20.63 | 0.00 | 5.17 |
| Binary Search | 0.79 | 35.71 | 5.17 | 50.00 |
| Tool-Caller | <u>19.05</u> | 45.24 | <u>18.97</u> | 53.45 |
| RAFFLES (Iterators) | 34.92 | <u>50.00</u> (63/126) | 25.86 | <u>58.62</u> (34/58) |

Table 9: Results of Mixtral-8x22b model on Who&When dataset

| Method | Algorithmically-Generated (126) | | Hand-Crafted (58) | |
|----------------------------|---------------------------------|--------------|-------------------|--------------|
| | Step-level | Agent-level | Step-level | Agent-level |
| Chat-LLM (One Shot) | <u>20.63</u> | <u>35.71</u> | 1.75 | <u>52.63</u> |
| Routers | | | | |
| Step by Step | 10.32 | 28.57 | 0.00 | 3.45 |
| Binary Search | 0.79 | 33.33 | <u>10.34</u> | 53.45 |
| Tool-Caller | 17.46 | 31.75 | 7.02 | 38.60 |
| RAFFLES (Iterators) | 29.37 | 47.62 | 15.79 | 38.60 |

D Extended Analysis

This section provides a detailed analysis of the convergence of RAFFLES over multiple iterations. Figure 5 illustrates the model’s convergence from iteration 1 to 3 on the Who&When dataset using Llama 3.3 70B. At an aggregate level, the predicted steps for both the Algorithmically-Generated and Hand-Crafted datasets converge toward the ground truth distribution. Focusing on the Hand-Crafted dataset in Figure 5b, the model initially tends to predict the decisive fault at an early step ($\hat{t} \leq 5$), resulting in a prominent spike in the first iteration’s distribution. Over subsequent iterations, this

Table 10: Results of gpt-oss-20b model on Who&When dataset

| Method | Algorithmically-Generated (126) | | Hand-Crafted (58) | |
|----------------------------|---------------------------------|--------------|-------------------|--------------|
| | Step-level | Agent-level | Step-level | Agent-level |
| Chat-LLM (One Shot) | 14.29 | 63.49 | 3.45 | 25.86 |
| Routers | | | | |
| Step by Step | 11.90 | 30.16 | 0.00 | 34.48 |
| Binary Search | 23.81 | 46.83 | 13.79 | 55.17 |
| Tool-Caller | <u>29.37</u> | 41.27 | <u>17.24</u> | 39.66 |
| RAFFLES (Iterators) | 44.44 | <u>58.73</u> | 22.41 | <u>53.45</u> |

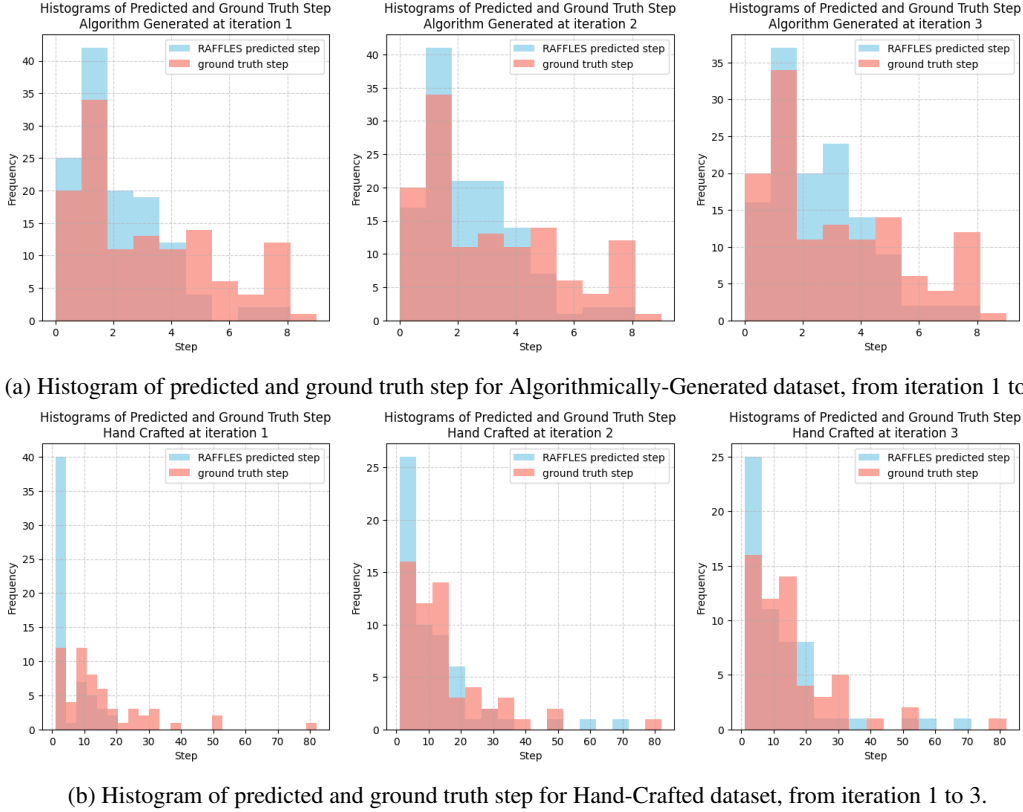


Figure 5: Histogram of predicted and ground truth step for Who&When dataset, from iteration 1 to 3 for Llama 3.3 70b model. Notice that, for a given iteration k , we do not show the data points where convergence already happened in prior steps $k' < k$, hence the difference in ground truth histogram.

distribution converges towards that of the ground truth, showcasing the efficacy of the iterative process.

The primary distinction between the Algorithmically-Generated (Figure 5a) and Hand-Crafted (Figure 5b) datasets lies in their ground truth distributions. Most ground truth steps in the Algorithmically-Generated dataset are within the first four steps, while the Hand-Crafted dataset has a more distributed range of steps. The LLM’s initial predictions often favor these early steps, a bias evident in Figure 5b. We hypothesize this is analogous to the "lost in the middle" phenomenon reported for LLMs in long-context scenarios [5]. Furthermore, the "first mistake" framing in the prompt may predispose the model to focus on the initial part of the trajectory, further necessitating an iterative process with the "correction prompt" within the 3rd Evaluator’s instructions. Beginning in iterations 2 and 3, critiques from this Evaluator prompt the Judge to reason more deeply and select candidates in later steps, thereby refining the prediction distribution to more closely match the ground truth.

Table 11: Percentage of changed agent-step pair (i, t) between current and prior iterations

| | Iteration 1→2 | Iteration 2→3 | Iteration 3→4 | Iteration 4→5 |
|----------------------------------|---------------|---------------|---------------|---------------|
| Algorithmically-Generated | 25.40% | 14.29% | 3.97% | 5.56% |
| Hand-Crafted | 37.93% | 8.62% | 3.45% | 5.17% |

This initial prediction bias, combined with the inherently larger search space of the Hand-Crafted problems, increases the reasoning difficulty. This confluence of factors contributes to the accuracy degradation observed in Figure 2b. Furthermore, we analyze the rate of convergence by computing the percentage of changed steps between consecutive iterations, presented in Table 11. While more than a quarter of the steps are revised between the first and second iterations, the rate of change decreases substantially by iteration 4. This pattern indicates diminishing returns, a characteristic also observed in prior research on self-refinement methods [34].

E Limitations and Next Step

A significant constraint of this study was the limitation on available computational resources. This precluded the use of more powerful, state-of-the-art large language models, such as those in the GPT and Claude series, for our experiments. These models often demonstrate enhanced capabilities in complex reasoning and generation tasks. Despite this, it is noteworthy that our findings are comparable to the benchmark results reported in the Who&When paper [7], suggesting the robustness of our approach even with more modest computational architectures. Future work would benefit from validating these results on larger-scale models to explore the upper bounds of performance.

A broader, systemic challenge confronting this domain is the pronounced scarcity of large-scale, high-quality data. In our experiments, we tested the publicly available Who&When dataset. The current fault attribution data landscape is insufficient for the comprehensive validation of the methods proposed in the fault attribution field. For future work, it’s important to curate larger, more diverse public datasets that can support more rigorous and generalizable model evaluation and training. Additionally, the development of reliable methods for generating synthetic data specifically tailored for fault attribution tasks would also be of great interest for the future.

Our work is also limited by the token context length budget. While adequate for the Who&When dataset, this is insufficient for more complex benchmarks like TRAIL [8], which features average context lengths of 300k to 700k tokens. Such extremely long contexts pose significant challenges for both the long-horizon reasoning capabilities of language models and the efficiency of model inference, demanding future innovations in model architecture and computational optimization.

F Prompt

F.1 Baseline prompt

We took the one-shot prompt in [7] and slightly modified the prompt to the Llama model chat template format.

Chat-LLM Prompt

You are an AI assistant tasked with analyzing a multi-agent conversation history when solving a real world problem.
The problem is: {problem}.
Identify which agent made an error, at which step, and explain the reason for the error.
Here’s the conversation: {failure_log}
Based on this conversation, please predict the following:
1. The name of the agent who made a mistake that should be directly responsible for the wrong solution to the real world problem. If there are no agents that make obvious mistakes,


```

decide one single agent in your mind. Directly
output the name of the Expert.
2. In which step the mistake agent first made mistake. For example,
in a conversation structured as follows:
{{
  "agent a": "xx",
  "agent b": "xxxx",
  "agent c": "xxxxx",
  "agent a": "xxxxxxx"
}},
each entry represents a 'step' where an agent provides input.
The 'x' symbolizes the speech of each agent. If the
mistake is in agent c's speech, the step number is 2. If the second
speech by 'agent a' contains the mistake, the step
number is 3, and so on. Please determine the step number where the
first mistake occurred.
3. The reason for your prediction. Please answer in the format.
Notice that you can *ONLY* output the json.
```json
{
 "agent_name": "",
 "step_number": 0,
 "reason_for_mistake": ""
}
```

```

For both the Step by Step baseline and binary search baseline, we use the following prompt template. We then use the specific task description to describe the required logic for Step by Step. These prompts are taken mostly from the Who&When paper with slight modification [7].

Prompt Template

You are an intelligent assistant that takes in a task description, and task output and complete based on requirements.

Task Description
{task_description}

Input Metadata
{input_metadata}

Task Output
{task_output}

Remember, that your output should only be a json and nothing else.

Step by Step instruction

You are an AI assistant tasked with evaluating the correctness of each step in an ongoing multi-agent conversation aimed at solving a real-world problem. Based on the conversation history in "history_up_to_step" up to the current step. Your task is to **determine whether the most recent agent's action contains an error that could hinder the problemsolving process**. Please respond with 'Yes' or 'No' and provide a clear explanation for your judgment. Note: Please avoid being overly critical in your evaluation.

Attention: Respond in the format:

1. Yes/No. Yes being that the pipeline failure is because of this latest step and No being that the pipeline failure is not because of this step.
2. Reason for the judgment.

Remember that you are not trying to answer the question based on the question given. Your job is to ****determine whether the most recent agent's action contains an error that could hinder the problemsolving process****.

Step by Step task output

Please answer in the format:

```
```json
{
 "judgement": "upper half" or "lower half",
 "reason": ""
}
```

#### Binary Search instruction

You are an AI assistant tasked with analyzing a segment of a multi-agent conversation. Multiple agents are collaborating to address a user query, with the goal of resolving the query through their collective dialogue. Your primary task is to identify location of the most critical mistake, and determine the single step in the conversation where this error occurs, ultimately leading to the failure in resolving the user's query.

Based on your analysis, predict whether the error is more likely to be located in the upper or lower half of the segment.

lower half is defined as the range lower half range and upper half is defined as the range upper half range.

Please simply output either 'upper half' or 'lower half'.

You should not output anything else.

#### Binary Search task output

Please answer in the format:

```
```json
{
  "judgement": "upper half" or "lower half",
  "reason": ""
}
```

F.2 Tool-Caller prompt

Our Tool-Caller mechanism is implemented using LangChain⁵ and is designed to improve upon traditional Step by Step methods. It features a **Planner** that leverages global access to the entire log to intelligently select a specific step to inspect via its id. A **Judge**, similar to the one in the Step by Step approach, then evaluates the process up to the step designated by the Planner.

This architecture allows the Tool-Caller to dynamically select the most relevant point for evaluation, breaking free from a predefined, rigid iterative sequence. Because the Planner has a global view of the process, its selection of the next candidate to inspect is significantly more efficient and intelligent.

⁵<https://www.langchain.com/>

Despite having a planner and judge like RAFFLES, the Tool-Caller method does not necessarily have a structured and iterative reasoning process. The planner does not reason explicitly before giving out the next candidate to check. There is only one judge, which is insufficient to provide enough reasoning to support the complexity of the fault attribution task.

Tool-Caller Prompt

```
You are an expert in planning
and calling agents to evaluate the input.
You are given a system log and a set of possible agents.
Based on the log, you will need to make one or more agent calls to
achieve the purpose.
If none of the agents can be used, point it out. If the given question
lacks the parameters required by the function, also point it out.
If you decide to invoke any of the function(s), you MUST put it in the
format of <agent>agent_name(args, kwargs)</agent>
You SHOULD NOT include any other text in the response. You should only
call each agent ONCE.
Here is a list of agents in JSON format that you can invoke.
1. You don't have to look over the entire conversation history one
after another, it is okay to choose the most important one first.
2. There is only one mistake in each conversation history, so you only
need to find one agent and one step.
3. Your goal is to find the agent at fault with the least number of agent
calls. So choose the agent to inspect wisely based on which agent looks
like that it has fault.
4. User cannot make a mistake, so there's no need to consider user input.
5. If you cannot find any mistake, you can output "no mistake" as the
agent name and -1 as the step number.
6. You should limit your tool calling to less than 3 times.
<|eot_id|><|start_header_id|>user<|end_header_id|>

Your job is to use agents to give a assessment score on each of the
following components of the complex system,
'''
Evaluate based on the following log of the pipeline:
{input_data['metadata']}
Now, start your evaluation. Your generation can only be of 2 of these
options.
Option A, if you want to call an agent,
1. **ONLY** output the tool calling and nothing else, such as
<agent>agent_name(args, kwargs)</agent>.
2. You can only call one agent at a time.
Option B, if you feel confident about the tools already used and is
ready to
give the overall score, provide the following information in json
Please answer in the format:
'''json
{{
  "agent_name": "",
  "step_number": 0,
  "reason_for_mistake": ""
}}
'''
```

Tool-Caller Judge Prompt

You are an AI assistant tasked with evaluating the correctness of each step in an ongoing multi-agent conversation aimed at solving a real-world problem.

Based on the conversation history in "history_up_to_step" up to the current step.

Your task is to **determine whether the most recent agent's action contains an error that could hinder the problemsolving process**. Please respond with 'Yes' or 'No' and provide a clear explanation for your judgment.

Note: Please avoid being overly critical in your evaluation.

Attention: Respond in the format:

1. Yes/No. Yes being that the pipeline failure is because of this latest step and No being that the pipeline failure is not because of this step.
2. Reason for the judgment.

Remember that you are not trying to answer the question based on the question given. Your job is to **determine whether the most recent agent's action contains an error that could hinder the problemsolving process**.

Please answer in the format:

```
```json
{
 "judgement": "yes" or "no",
 "reason": ""
}
```

```
conversation history:
{prompt_history}
```

### E.3 RAFFLES Prompt

RAFFLES's judge prompt uses the same prompt template as the Step by Step prompt template.

#### Judge Prompt

You are an AI assistant tasked with analyzing a multi-agent conversation history when solving a real world problem. Identify which agent made an error, at which step, and explain the reason for the error.

Based on this conversation, please predict the following:

1. The name of the agent who made a mistake that should be directly responsible for the wrong solution to the real world problem. If there are no agents that make obvious mistakes, decide one single agent in your mind. Directly output the name of the Expert.
2. In which step the mistake agent first made mistake.

**\*\*You must always output an agent name and a step number.\*\*** Null, None, or empty values are strictly forbidden for the "agent\_name" and "step\_number" fields.

Notice that you should point out the agent and the step such that all three of the following criteria are satisfied:

1. The agent made a mistake at that step.
2. It is the first mistake step that relates to the final wrong outcome.
3. The mistake was not corrected by or correctable by later agents.

**## Handling Ambiguity (Fallback Procedure)**

In cases where no single agent or step perfectly meets all three criteria (for example, if the error was collaborative or no obvious mistake exists), **\*\*you must apply the following logic to make a**

determination:\*\*

Identify the agent whose contribution was the **most pivotal** in setting the final, incorrect direction. This could be the agent who introduced the flawed method, provided the key piece of wrong information, or signed off on the solution without a final critical review. Select the corresponding step. This ensures you always provide a "best guess" even in unclear situations.

#### task output format

Please answer in the format:

```
““json
{
 "agent_name": "The name of the faulty agent you identified,
satisfying all the three criteria.",
 "step_number": "The step number where the chosen agent made
the mistake, satisfying all the three criteria.",
 "mistake_reason": "Briefly explain why the agent made a
mistake at that step. Reference the log as needed for clarity.",
 "first_mistake": "Briefly explain why it is the first mistake
step that relates to the final wrong outcome. Reference the
log as needed for clarity.",
 "mistake_not_corrected": "Briefly explain how the mistake was
not corrected by or correctable by later agents. Reference the
log as needed for clarity."
}
““
```

#### Evaluator 1: Mistake Prompt

You are a rigorous and meticulous logic verifier, serving as a critical component within a reasoning system dedicated to fault attribution in complex system logs. Your specific assigned task is to verify the reasoning logic provided by your partner. Your sole purpose is to identify flaws, inconsistencies, and leaps in logic, and you must not be swayed by your partner's conclusion, but only by the soundness of their argument. Your partner will identify the agent and the step such that all three of the following criteria are satisfied:

1. The agent made a mistake at that step.
2. It is the first mistake step that relates to the final wrong outcome.
3. The mistake was not corrected by or correctable by later agents.

Hence, you are provided with the following inputs:

- Task Log: A multi-agent conversation log
- Error Step: Output from your partner with candidate point of fault and their associated reasoning.

Your task is **ONLY** to think whether the argument provided by your partner for 'correctly pointing out a faulty agent and step number' is logical. You will try to verify the argument from the task log and give your reasons about whether this argument is logical or not. Then, you will give a confidence score between 0 to 100 indicating your confidence in the soundness of your partner's argument.

For example, general or non-specific reasoning that cannot be verified by a non-expert is less logical than specific reasoning that can be easily verified. Further, if you are unable to verify the correctness of the argument from the task log, you should give a low confidence score.

## Task Log ##

```

{task_log}
Error Step
{error_step}
Your output format
You should directly output a json in the following format:
```json
{
  "reason": "Briefly explain why the given argument for 'correctly
pointing out a faulty agent and step number' is sound or unsound.
If unsound, identify the specific flaw.",
  "confidence": "Assign an integer score between 0 to 100 indicating
your confidence in the **soundness and logical consistency of the
partner's argument**. 100 means the argument is logical, specific,
and fully supported by the log. 0 means the argument is illogical,
non-specific, or contradicts the log."
}
}
```

```

### Evaluator 2: First Mistake Prompt

You are a rigorous and meticulous logic verifier, serving as a critical component within a reasoning system dedicated to fault attribution in complex system logs. Your specific assigned task is to verify the reasoning logic provided by your partner. Your sole purpose is to identify flaws, inconsistencies, and leaps in logic, and you must not be swayed by your partner's conclusion, but only by the soundness of their argument.

Your partner will identify the agent and the step such that all three of the following criteria are satisfied:

1. The agent made a mistake at that step.
2. It is the first mistake step that relates to the final wrong outcome.
3. The mistake was not corrected by or correctable by later agents.

Hence, you are provided with the following inputs:

- Task Log: A multi-agent conversation log
- Error Step: Output from your partner with candidate point of fault and their associated reasoning.

Your task is **ONLY** to think whether the argument provided by your partner for 'finding the first mistake in the pipeline' is logical. You will try to verify the argument from the task log and give your reasons about whether this argument is logical or not. Then, you will give a confidence score between 0 to 100 indicating your confidence in the soundness of your partner's argument.

For example, general or non-specific reasoning that cannot be verified by a non-expert is less logical than specific reasoning that can be easily verified. Further, if you are unable to verify the correctness of the argument from the task log, you should give a low confidence score.

## Task Log ##

```
{task_log}
```

## Error Step ##

```
{error_step}
```

## Your output format ##

You should directly output a json in the following format:

```

```json
{
  "reason": "Briefly explain why the given argument for 'finding the
first mistake in the pipeline' is sound or unsound. If unsound,

```

```

        identify the specific flaw.",
        "confidence": "Assign an integer score between 0 to 100 indicating
        your confidence in the **soundness and logical consistency of the
        partner's argument**. 100 means the argument is logical, specific,
        and fully supported by the log. 0 means the argument is illogical,
        non-specific, or contradicts the log."
    }}
    ""

```

Evaluator 3: Correction Prompt

You are a rigorous and meticulous logic verifier, serving as a critical component within a reasoning system dedicated to fault attribution in complex system logs. Your specific assigned task is to verify the reasoning logic provided by your partner. Your sole purpose is to identify flaws, inconsistencies, and leaps in logic, and you must not be swayed by your partner's conclusion, but only by the soundness of their argument.

Your partner will identify the agent and the step such that all three of the following criteria are satisfied:

1. The agent made a mistake at that step.
2. It is the first mistake step that relates to the final wrong outcome.
3. The mistake was not corrected by or correctable by later agents.

Hence, you are provided with the following inputs:

- Task Log: A multi-agent conversation log
- Error Step: Output from your partner with candidate point of fault and their associated reasoning.

Your task is **ONLY** to think whether the argument provided by your partner for 'how this mistake was never corrected afterwards' is logical. You will try to verify the argument from the task log and give your reasons about whether this argument is logical or not. Then, you will give a confidence score between 0 to 100 indicating your confidence in the soundness of your partner's argument.

For example, general or non-specific reasoning that cannot be verified by a non-expert is less logical than specific reasoning that can be easily verified. Further, if you are unable to verify the correctness of the argument from the task log, you should give a low confidence score.

```

## Task Log ##
{task_log}
## Error Step ##
{error_step}
## Your output format ##
You should directly output a json in the following format:
""json
{{
    "reason": "Briefly explain why the given argument for 'how this
    mistake was never corrected afterwards' is sound or unsound. If
    unsound, identify the specific flaw.",
    "confidence": "Assign an integer score between 0 to 100 indicating
    your confidence in the **soundness and logical consistency of the
    partner's argument**. 100 means the argument is logical, specific,
    and fully supported by the log. 0 means the argument is illogical,
    non-specific, or contradicts the log."
}}
""

```