RAFFLES: Reasoning-based Attribution of Faults for LLM Systems

Anonymous ACL submission

Abstract

The advent of complex, interconnected long-horizon LLM systems has made it incredibly tricky to identify where and when these systems break down. Evaluation capabilities that currently exist today are limited in that they often focus on simple metrics, end-to-end outcomes, and are dependent on the perspectives of humans. In order to match the increasing complexity of these many component systems, evaluation frameworks must also be able to reason, probe, iterate, and understand the nuanced logic passing through these systems. 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. We evaluated RAFFLES with several benchmarks - the Who&When dataset to identify step-level faults in agentic systems and the ReasonEval datasets to diagnose step-level mathematical reasoning errors. RAFFLES outperforms strong baselines, achieving an accuracy of over 20% and 50% on the Who&When Hand-Crafted and Algorithmically-Generated datasets, and over 80% on the ReasonEval datasets. 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 (Yao et al., 2023b), Toolformer (Schick et al., 2023), and Reflexion (Shinn et al., 2023) 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 (Epperson et al., 2025; Liu et al., 2023a; Xu et al., 2025). 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* (Epperson et al., 2025; Zhang et al., 2025a; Deshpande et al., 2025). 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 *when* they form.

Automatic failure attribution within LLM multi-component agentic systems is an emerging area of focus for the evaluation community (Zhang et al., 2025a; Zhou et al., 2025; Deshpande et al., 2025; Cemri et al., 2025). However, preliminary results have demonstrated limited success due to the challenges imposed by detecting faults in complex systems. For example, the LLM-as-a-judge has become a cornerstone of automated evaluation due to its flexibility in approximating human evaluators (Zheng et al., 2023; Fu et al., 2023), and yet single-pass LLM evaluators struggle to detect faults within the long trajectories of agentic systems (Zhang et al., 2025a; Deshpande et al., 2025). Effective evaluation by which one can pinpoint faulty planning, logic, tool calls, calculations, code, and more, within multi-component systems *requires* its own set of capabilities (e.g., reasoning, planning, tool calling, iterating). According to a framework from a 2025 ICML tutorial, LLM systems can be

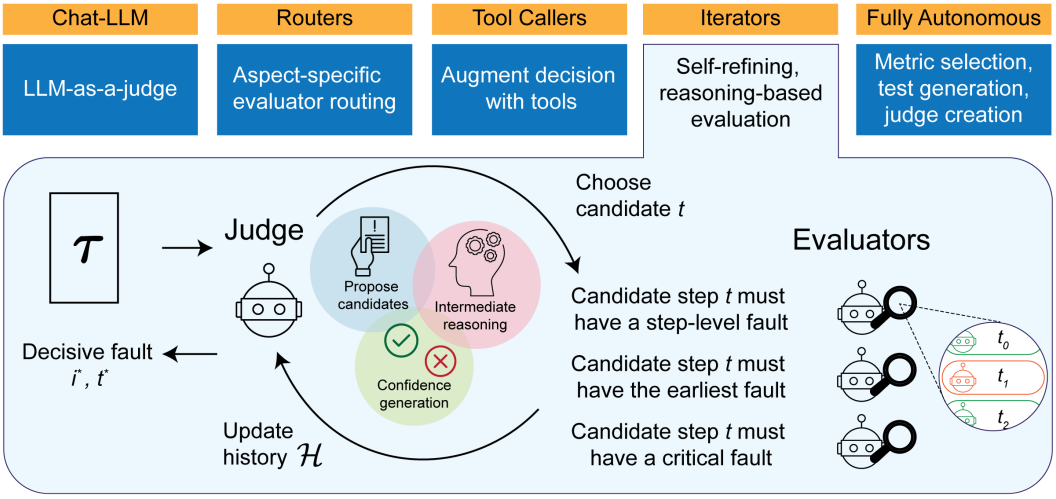


Figure 1: **Our proposed RAFFLES framework for multi-turn agentic evaluation by reasoning-based fault attribution.** RAFFLES leverages specialized Evaluators designed to assess candidate faults based on the criteria of a decisive fault. Each Evaluator takes in the full log τ and intermediate reasoning, which are passed to subsequent iterations until a decisive fault is determined.

grouped into five levels of increasing complexity: Chat-LLM, Routers, Tool-Callers, Iterators, and Fully Autonomous systems (Hassani et al., 2025). We believe that evaluation systems must likewise evolve in complexity in a parallel manner: from single judgment Chat-LLMs, to aspect-specific evaluator routing, evaluations 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-step 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 (Lin, 2004; Papineni et al., 2002). 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. Suppose we extend this system to include tools for retrieval and a reflection loop to self-correct those retrieved documents (i.e. Yue et al. (2024)); every additional component that is added increases the scope of agent activity, the interaction effects, and the multitude of decisions that need to be tracked 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-step LLM systems. First, we extend and structure the reasoning capabilities of our LLM-as-a-judge (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 (Zhang et al., 2025a), which consists of agentic logs and agent-step fault pairs derived from GAIA (Mialon et al., 2023) and AssistantBench (Yoran et al., 2024). Our experiments demonstrate that structured iterative reasoning provides a significant and robust performance advantage over existing methods, and across diverse model families. Using Claude Sonnet 4, RAFFLES improved the step-level accuracy on fault attribution from a previous (simulation-free) best of 36.20% to 51.59% on the Algorithmically-Generated dataset, and 18.20% to 27.59% on the Hand-Crafted dataset. We also demonstrated the versatility of RAFFLES on the ReasonEval benchmark (Xia et al., 2025), which contains step-level annotations of reasoning chains for Math problems. With Claude Sonnet 4, RAFFLES improved step-level accuracy from 73.58% on a strong one-shot baseline to 84.91% on MR-MATH-Invalid, and from 75.46% to 83.78% on MR-GSM8K-Original.

Results on ReasonEval help validate the effectiveness of our framework in pinpointing the initial point of failure in complex reasoning processes, a critical requirement as we scale to more and more complex multi-component systems.

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 (Fu et al., 2023; Li et al., 2023; Liu et al., 2023b; Zheng et al., 2023), and (b) recent advances in the reasoning capabilities of LLMs (Lightman et al., 2023; Wei et al., 2022; Wang et al., 2022; Yao et al., 2023a). Enhancements of the LLM-as-a-judge include new methods that incorporate planning within the prompt (Saha et al., 2025), self-check reasoning (Miao et al., 2023), debate (Chan et al., 2023), and juries as judges (Verga et al., 2024). 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 (Zhou et al., 2022; Khot et al., 2022), natural language confidence estimation (Lin et al., 2022; Xiong et al., 2023; Tian et al., 2023), as well as iterative refinement (Shinn et al., 2023; Madaan et al., 2023).

Fault attribution in agentic systems. Numerous evaluation frameworks (Zhuge et al., 2024; Zhang et al., 2025a; Deshpande et al., 2025; Chang et al., 2024; Liu et al., 2025; Pan et al., 2024) have shown preliminary success evaluating agentic benchmark datasets like GAIA, SWEbench, and AssistantBench (Mialon et al., 2023; Jimenez et al., 2023; Yoran et al., 2024). Frameworks that detect trajectory-breaking faults include (a) a multi-agent debugging tool (Epperson et al., 2025), (b) a prompt-customized LLM-as-a-judge implementation (Deshpande et al., 2025), and (c) methods that horse race against the LLM-as-a-judge, such as multi-pass binary search and Step by Step implementations (Zhang et al., 2025a). Following the release of the Who&When dataset (Zhang et al., 2025a), recent work falls into two main categories: (1) simulation-based methods that simulate partial or entire trajectories to determine faults (Ge et al., 2025), and (2) simulation-free methods (Yu et al., 2025; Ma et al., 2025), which our paper mainly focuses on.

Fault attribution in reasoning chains. With the growing popularity of chain-of-thought (Wei et al., 2022; Kojima et al., 2022) and reasoning models (e.g. Agarwal et al. (2025)), advancement in pinpointing errors within a model's reasoning chain has been propelled by two concurrent efforts: the development of benchmarks with fine-grained, step-level error annotations (Xia et al., 2025; Song et al., 2025; He et al., 2025; Zheng et al., 2024), and the proposal of novel detection methods. Novel methods include both training-free approaches (Hao et al., 2024) and fine-tuned Process Reward Models (Xia et al., 2025; She et al., 2025; Zhang et al., 2025b). We benchmark our framework on fault attribution within reasoning chains as a means to validate its performance and versatility.

3 Background

A full trajectory from an agentic system or a step-by-step reasoning trajectory τ can be written as $\tau = (\tau_0, \ldots, \tau_T)$ where T is a terminal time step. The success or failure of a trajectory is determined by a binary outcome function $Z(\tau)$:

$$Z(\tau) = \begin{cases} 1 & \text{if } \tau \text{ leads to the correct outcome,} \\ 0 & \text{otherwise.} \end{cases}$$

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 (Xie et al., 2024; Mialon et al., 2023). Given a dataset, we assume that there exists one or more τ^* such that $Z(\tau^*) = 1$. When a system ultimately fails ($Z(\tau) = 0$), we seek to understand where and when the failure occurs within the trajectory. Even when the final outcome is correct, there is still interest in understanding if and when there are errors in the reasoning process (Xia et al., 2025).

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

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

The function f is usually a case-by-case LLM-judge that follows a certain taxonomy, most commonly hallucination or execution error (Deshpande et al., 2025; Cemri et al., 2025).

Definition 3.2 (First Step-Level Fault Attribution) *We define the task of First Step-Level Fault Attribution as finding the first error that occurs in a multi-step LLM system. Specifically,*

$$t^* = \arg\min_t \quad t$$
$$\textit{subject to} \quad (a)\ f(\tau_t) > \varepsilon$$

Definition 3.2 is helpful in cases where researchers want to identify the first failure step of the trajectory, such as when analyzing mathematical reasoning trajectories in ReasonEval (Xia et al., 2025).

However, in cases where the trajectory ends up in a failure ($Z(\tau) = 0$), a step-level error may or may not be directly responsible for the final failure. To establish this relationship, 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 concepts.

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

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

Here $\tau | a_t := \tilde{a}_t$ denotes the alternative trajectory that unfolds after the intervention at time t. A fault is trivial when it was minor, corrected later, or if a subsequent, unavoidable fault later in the trajectory would have caused a failure regardless. Unlike a trivial fault, which has no bearing on the final outcome of a trajectory, we now define a critical fault that directly impacts the result.

Definition 3.4 (Critical Fault) *Let τ be a trajectory with a failed final outcome ($Z(\tau) = 0$) containing a step-level fault a_t such that an alternative that replaces a_t with $\tilde{a}_t$ results in a successful trajectory. Formally, a_t is a critical fault if*

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

This definition isolates actions that were decisively responsible for the failure.

Based on these concepts, we finally define a **Decisive Fault** as the *first critical 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 (Zhang et al., 2025a), 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 3.5 (Decisive Fault Attribution) *The decisive fault is the first critical fault to occur in a trajectory. The process of identifying it can be formulated as a constrained optimization problem:*

$$t^* = \arg\min_t \quad t$$
$$\textit{subject to} \quad (a)\ f(\tau_t) > \varepsilon$$
$$(b)\ Z(\tau | a_t := \tilde{a}_t) = 1, \exists \tilde{a}_t \in \mathcal{A}$$

Definition 3.5 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 trajectory τ, we designed RAFFLES (Algorithm 1) as a Judge-Evaluator iterative LLM system to attribute faults based on Definition 3.5, which we distill into the following three necessary criteria:

1. **Fault Condition**. The action of agent i at step t must constitute a step-level fault, satisfying Definition 3.1.

2. **Primacy**. The action of agent i at step t must be the earliest critical fault, satisfying the goal of finding the $\min$ of t in Definition 3.2.

3. **Decisiveness**. The agent i's mistake must be a critical fault, satisfying constraint (b) in Definition 3.4.

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 candidate step 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. The 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 E.3.

Evaluator: Each Evaluator E_p, for $p \in \{1, 2, 3\}$ evaluates one of the three criteria for decisive faults given the proposed step 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 to produce the likelihood of error in Definition 3.1. We also designed an additional rule-based Evaluator $p = 4$ to validate whether the proposed candidate step t 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 the Evaluator prompt in Appendix E.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 step t selection strategy.

The iteration concludes when either (a) TERMINATE: C is greater than a threshold of 350 after considering the log τ and the evidence in $\mathcal{H}$, or (b) when a predefined maximum number of iterations K is reached and we take the candidate with the highest confidence in $\mathcal{H}$. The step t that has the highest confidence C in history $\mathcal{H}$ after this iteration is then considered the decisive fault t^*. If the Judge determines that there is no decisive fault, and the confidence from the Evaluators reaches the threshold, then $t^* =$ None signals an absence of decisive fault. An illustrative example of the iterative process can be found in Appendix F.

Overall, RAFFLES relies on the iterative and structured reasoning process between the Judge and the Evaluators, tailored to solve the constraint optimization problem defined in Definition 3.5. 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. In addition, RAFFLES prompts are task-agnostic; adding a new dataset only requires minor revisions to the problem statement. Details of our prompts can be found in Appendix E.3.

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 $k \leq K$ **do**
4: $t, R_j \leftarrow \text{Judge}(\tau, \mathcal{H})$ {Propose fault step and rationale from τ}
5: $C, R_e \leftarrow \text{Evaluator}(t, R_j, \tau)$ {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: **return** t^*, the decisive fault identified from $\mathcal{H}$

5 Experiments

5.1 Baseline Methods

We perform comprehensive experiments on RAFFLES with open-source and proprietary models of various model sizes and families, including Llama-3.3-70B-Instruct (Meta, 2024), gpt-oss-20b model (OpenAI, 2025), and claude-sonnet-4-20250514 (Anthropic, 2025), with results for additional models in Appendix C. All of the open-source models are deployed on AWS p4de nodes with NVIDIA A100 80GB GPUs using vLLM (Kwon et al., 2023) without quantization. For all experiments except Claude, a greedy search is used with a maximum context length of 128k tokens; Claude uses a maximum context length of 200K tokens.

In addition to RAFFLES, we implement four baselines categorized by the evaluation system hierarchy shown in Figure 1, covering prior work as well as a strong Tool-Caller baseline that we introduce. The first baseline is a one-shot **LLM-as-a-judge** from (Zhang et al., 2025a), which directly evaluates the entire log and problem statement to find the faulty step. The second class of baseline evaluators include LLM routers proposed in Who&When (Zhang et al., 2025a), including a **Step by Step** method, which examines the log sequentially and stopping only 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 introduce a **Tool-Caller** baseline 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 E.

5.2 Dataset

Our primary experiments are conducted on the **Who&When** dataset (Zhang et al., 2025a) without ground truth; a benchmark specifically created for fault attribution in multi-agent systems. We utilize its two subsets to evaluate performance across different agentic setups: (a) the Algorithmically-Generated subset provides breadth with 191 unique agents across 126 logs (avg. 9 steps), and (b) the Hand-Crafted subset offers depth with 5 agents across 58 logs and an avg. of 51 steps.

To demonstrate RAFFLES' versatility, we tested fault attribution on mathematical reasoning using two datasets from **ReasonEval** (Xia et al., 2025). The first dataset, MR-MATH-Invalid, contains 159 trajectories with correct output (avg. 9 steps), of which 83 have an incorrect reasoning step. The second dataset, MR-GSM8K, consists of 2,777 samples with question types that are "original" (avg. 7.05 steps) and "reversed" (avg. 11.26 steps). For ReasonEval tasks, the concept of a critical fault is not applicable based on how the dataset was annotated. Thus, to adapt RAFFLES for these datasets, we omit the Decisiveness criterion (Section 4) from our framework while the other components remain unchanged. A detailed discussion of our dataset selection criteria, statistics, and potential limitations is available in Appendix A.

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 s steps. A prediction $\hat{t}$ is considered correct if it falls within s steps of the ground-truth fault step (i.e. $|\hat{t} - t_{gt}| \leq s$). This metric is a good indicator of practical utility, as pinpointing a fault to a small window (e.g., $s \leq 2$) is often sufficient to guide efficient manual inspection and debugging of interacting components. We discuss alternative metrics for each dataset in Appendix B, and show experiment results using alternative metrics in Appendix C.

6 Results

6.1 RAFFLES Results on Who&When

Table 1 details the step-level accuracy on the Algorithmically-Generated and Hand-Crafted datasets, using a maximum of $K = 2$ iterations.

RAFFLES detects decisive faults better than baseline methods across diverse model families. On the Algorithmically-Generated dataset, RAFFLES outperforms our strongest baselines, Tool-Caller, by 31% with Llama 3.3 70B and Binary Search, by 55% with Claude Sonnet 4. The performance superiority is further validated on the more challenging Hand-Crafted dataset, where RAFFLES surpasses Tool-Caller by 53% (Llama 3.3 70B). Furthermore, as shown in Table 3, RAFFLES consistently outperforms our Chat-LLM baseline across nearly all tolerance thresholds in both datasets. Together with the results of additional models in Appendix C, we show that RAFFLES provides a universal performance lift across all benchmark models, from large-scale Mixture-of-Experts (MoE) models with pre-trained reasoning capabilities to smaller models with limited pre-trained reasoning.

Fault attribution methods using partial trajectories are insufficient for robust performance improvement. Performance with Step by Step appears to be highly dependent on the pretrained model (Table 1) as powerful proprietary models overcome incomplete information, while other (open) models cannot. With a history limited to prior context, Step by Step fails on long trajectories like those in the Hand-Crafted dataset; e.g., accuracy drops from 6.9 to 3.45 for Llama 3.3 70B. The Binary Search method is moderately effective for long contexts, but less so for shorter ones; the greatest lift is for Claude-4-Sonnet on the Hand-Crafted dataset. Overall, these results highlight the need for a system that combines a global, end-to-end perspective with focused, local analysis. RAFFLES consistently outperforms these baselines by embodying these principles; the Judge operates on the global context to form hypotheses, and the Evaluators focus on the reasoning process for the hypothesized fault step.

Table 1: Step-level accuracy on Who&When datasets. Bold text highlights the best performance, and underlined text represents the second best. Alg. represents Algorithmically-Generated; Hand. represents Hand-Crafted dataset.

	Llama 3.3 70B		gpt-oss-20b		Claude Sonnet 4	
Method	**Alg.**	**Hand.**	**Alg.**	**Hand.**	**Alg.**	**Hand.**
Chat-LLM	19.05	6.90	14.29	3.45	29.37	3.45
Routers (Step by Step)	6.35	3.45	19.84	**22.41**	30.16	12.07
Routers (Binary Search)	4.76	10.34	15.08	15.52	33.33	20.69
Tool-Caller	33.33	13.56	29.37	17.24	30.95	18.97
RAFFLES	**43.65**	**20.69**	**44.44**	**22.41**	**51.59**	**22.41**

Table 2: Step-level accuracy on ReasonEval datasets. Baseline Chat-LLM is based on the prompt from Xia et al. (2025). Bold text highlights the best performance, and underlined text represents the second best. Math represents MR-MATH-Invalid dataset, gsm-o represents MR-GSM8K-Original, and gsm-r represents MR-GSM8K-Reversed.

	Llama 3.3 70B			gpt-oss-20b			Claude Sonnet 4		
Method	**math**	**gsm-o**	**gsm-r**	**math**	**gsm-o**	**gsm-r**	**math**	**gsm-o**	**gsm-r**
Chat-LLM	65.41	69.75	61.07	84.28	84.77	**78.44**	73.58	75.46	67.77
RAFFLES $K = 0$	68.55	74.68	**62.18**	83.02	84.84	76.60	82.39	82.65	70.42
RAFFLES $K = 1$	**69.18**	75.81	61.81	84.28	**85.12**	77.26	79.87	83.00	74.83
RAFFLES $K = 2$	68.55	**76.02**	61.95	**84.91**	**85.12**	77.41	**84.91**	**83.78**	**75.28**

Structured reasoning is more effective than flexible tool calling in fault attribution. While Table 1 shows that 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.

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 Figure 2 reveals, 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.

RAFFLES establishes a new state of the art on the Who&When dataset (Table 4). RAFFLES significantly outperforms the best simulation-free method by 35% (Algorithmically-Generated) and 52% (Hand-Crafted). More importantly, it also surpasses leading simulation-based methods by 8.7%

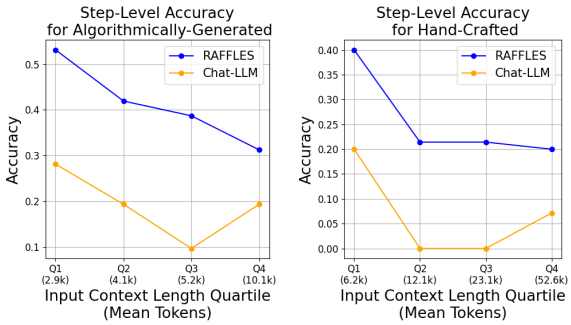


Figure 2: Llama 3.3 70B performance degrades with input trajectory length. Quartile of token length is with respect to the corresponding dataset.

on the Algorithmically-Generated dataset. Our results demonstrate that structured iterative reasoning can supplant the need for costly simulations, particularly in shorter-horizon scenarios. In Appendix C, we show results on agent-level metrics, where RAFFLES maintains superior performance across all datasets.

6.2 RAFFLES Results on ReasonEval

RAFFLES demonstrates strong generalization to fault attribution in reasoning chains. A key advantage of RAFFLES is task-agnostic structured reasoning. With minimal prompt changes from agentic fault attribution, the results on the Rea-

Table 3: Tolerant step-level accuracy with $\pm s$ 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**

Table 4: State-of-the-art step-level accuracy results on the Who&When dataset without ground truth, as reported in recent literature. Bold text shows overall best performance, and underline highlights best performance per category.

Category	**Methods**	**Model**	**Algo.**	**Hand.**
Requires Simulation	FAMAS (Ge et al., 2025)	Qwen2.5-72B	<u>23.81</u>	<u>**41.38**</u>
No Simulation	CORRECT (Yu et al., 2025)	GPT-5	38.10	17.20
	Causal Framework (Ma et al., 2025)	N/A	36.20	18.20
	RAFFLES (Ours, $K = 5$)	Claude Sonnet 4	<u>**51.59**</u>	<u>27.59</u>

sonEval dataset, presented in Table 2, show that RAFFLES (with $K = 2$) significantly outperforms the LLM-judge baseline. When implemented with the Claude Sonnet 4 model, our method improves upon the baseline by 15.40% on MR-MATH-Invalid, 11.08% on MR-GSM8K-Reversed, and 11.03% on MR-GSM8K-Original. Even with no iteration $K = 0$, a single-pass with a structured LLM judge, Claude-based RAFFLES achieves a step-level accuracy of 82.39% (+12%) on MR-MATH-Invalid set and 82.65% (+9.5%) on MR-GSM8K-Original dataset, outperforming the original prompt with simple chain-of-thoughts. The consistent results across all model families demonstrate that structured reasoning is inherently more effective. Moreover, as detailed in Appendix C, this performance surpasses previously reported results from models trained on the same dataset. These results validate the effectiveness of RAFFLES in pinpointing the initial point of failure in complex reasoning processes, a critical requirement as we scale to more complex multi-component systems.

Iterative reasoning improves fault attribution performance. Table 2 shows RAFFLES performance when choosing different max step level K. We observed that iterative reasoning improves step-level accuracy for the three models at $K = 2$ for almost all datasets. We found that (1) iterative reasoning helps LLMs narrow in on the portion of the trajectory most likely to contain faults, as evidenced by the convergence of the target and predicted distributions for fault steps in Figure 4; (2) the efficacy of iterative reasoning is affected by the length of trajectory (Figure 3), especially when the decisive fault is located in the middle of a trajectory, as observed in Liu et al. (2023a); (3) iterative reasoning improvements are not always monotonic, echoing the findings of Madaan et al. (2023), necessitating the design of high-confidence early stop criteria and a maximum step size K (Table 2). In effect, our experiments have shown that models benefit from structured iterative reasoning during fault attribution, and this has enabled our framework to outperform strong baselines. Further analysis on this topic can be found in Appendix D.

7 Conclusion

We introduced RAFFLES, a novel iterative evaluation architecture designed to systematically identify decisive faults in agentic systems. 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 solved in a single step, our results show a step-level accuracy of over 51% on the Algorithmically-Generated (compared to published simulation-free best of 36%) and over 27% on the Hand-Crafted dataset (versus best of 18%). Finally, we demonstrated RAFFLES' versatility on fault attribution in math reasoning trajectories, showing 15.4% improvement over baseline.

Limitations

A primary, systemic limitation confronting the field is the pronounced scarcity of large-scale, high-quality datasets for agentic fault attribution. Our experiments were conducted on the publicly available Who&When dataset, but the current data landscape is insufficient for the comprehensive validation and generalization of emerging methods. A critical direction for future research is the curation of larger and more diverse public benchmarks that can support more rigorous model evaluation. Concurrently, the development of reliable methodologies for generating high-fidelity synthetic data tailored for fault attribution tasks would represent a significant contribution to advancing the field.

While we demonstrated the applicability of RAFFLES to the ReasonEval dataset, our primary focus was not on creating a superior process reward model. This scope presents a clear avenue for future investigation: applying our methodology more broadly to fault attribution for reasoning models and systematically evaluating its performance against specialized reward-based approaches. Although the iterative reasoning employed by RAFFLES incurs higher latency than a fine-tuned process reward model, it remains highly viable for less time-sensitive applications. These include the in-depth analysis of reasoning processes for model evaluation and the generation of high-quality synthetic datasets for training more efficient downstream models.

Ethical Considerations

This work has undergone a comprehensive internal risk evaluation. We have verified that all prior works, datasets, and models are properly cited and used in accordance with their licenses and intended purposes. All datasets were anonymized, and detailed statistics and descriptions can be found in the original source papers and in Appendix A. The scope of this study is limited to English-language datasets with a focus on agentic systems and mathematical problems. We utilized an AI assistant for grammar checking, LaTeX formatting, and fluency improvements.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Anthropic. 2025. Introducing the claude 4 family. https://www.anthropic.com/news/claude-4-family. Accessed: 2025-10-06.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

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

Will Epperson, Gagan Bansal, Victor C Dibia, Adam Fourney, Jack Gerrits, Erkang Zhu, and Saleema Amershi. 2025. Interactive debugging and steering of multi-agent ai systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–15.

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. Gptscore: Evaluate as you desire. *arXiv preprint arXiv:2302.04166*.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. 2025. Who is introducing the failure? automatically attributing failures of multi-agent systems via spectrum analysis. *arXiv preprint arXiv:2509.13782*.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, and 1 others. 2024. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*.

Hamed Hassani, Amin Karbasi, and Alex Robey. 2025. Jailbreaking llms and agentic systems: Attacks, defenses, and evaluations. Tutorial presented at ICML, Vancouver, CA.

Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Zhongyuan Peng, Zhaoxiang Zhang, Zhicheng Zheng, Wenbo Su, and 1 others. 2025. Can large language models detect errors in long chain-of-thought reasoning? *arXiv preprint arXiv:2502.19361*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpacaeval: An automatic evaluator of instruction-following models.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023a. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*.

Zhiwei Liu, Jielin Qiu, Shiyu Wang, Jianguo Zhang, Zuxin Liu, Roshan Ram, Haolin Chen, Weiran Yao, Huan Wang, Shelby Heinecke, and 1 others. 2025. Mcpeval: Automatic mcp-based deep evaluation for ai agent models. *arXiv preprint arXiv:2507.12806*.

Guoqing Ma, Jia Zhu, Hanghui Guo, Weijie Shi, Jiawei Shen, Jingjiang Liu, and Yidan Liang. 2025. Automatic failure attribution and critical step prediction method for multi-agent systems based on causal inference. *arXiv preprint arXiv:2509.08682*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Meta. 2024. Llama-3.3-70b-instruct. https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct. Accessed: 2025-10-06.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Ning Miao, Yee Whye Teh, and Tom Rainforth. 2023. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*.

OpenAI. 2025. gpt-oss-120b gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Swarnadeep Saha, Xian Li, Marjan Ghazvininejad, Jason Weston, and Tianlu Wang. 2025. Learning to plan & reason for evaluation with thinking-llm-as-a-judge. *arXiv preprint arXiv:2501.18099*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Shuaijie She, Junxiao Liu, Yifeng Liu, Jiajun Chen, Xin Huang, and Shujian Huang. 2025. R-prm: Reasoning-driven process reward modeling. *Preprint*, arXiv:2503.21295.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *arXiv preprint arXiv:2305.14975*.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. 2025. Evaluating mathematical reasoning beyond accuracy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 27723–27730.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2023. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*.

Austin Xu, Srijan Bansal, Yifei Ming, Semih Yavuz, and Shafiq Joty. 2025. Does context matter? contextual-judgebench for evaluating llm-based judges in contextual settings. *arXiv preprint arXiv:2503.15620*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

Ori Yoran, Samuel Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968.

Yifan Yu, Moyan Li, Shaoyuan Xu, Jinmiao Fu, Xinhai Hou, Fan Lai, and Bryan Wang. 2025. Correct: Condensed error recognition via knowledge transfer in multi-agent systems. *arXiv preprint arXiv:2509.24088*.

Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. 2024. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and 1 others. 2025a. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Jingwen Zhou, Jieshan Chen, Qinghua Lu, Dehai Zhao, and Liming Zhu. 2025. Shielda: Structured handling of exceptions in llm-driven agentic workflows. *arXiv preprint arXiv:2508.07935*.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, and 1 others. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*.

Table 5: Basic statistics of the Who&When and ReasonEval datasets.

Dataset	Subset	Sample Size	Avg. Steps	Avg. Words
Who&When	Algorithmically-Generated	126	8.72	1,507.33
	Hand-Crafted	58	50.60	7,459.91
ReasonEval	MR-MATH-Invalid	159	6.78	179.91
	MR-GSM8K-original	1,418	7.05	104.82
	MR-GSM8K-reversed	1,359	11.26	164.98

Appendix

A Discussion on Datasets

As detailed in Table 5, a statistical comparison reveals that the Who&When dataset presents a substantially more complex challenge than ReasonEval. The primary distinction lies in the trajectory length; all three ReasonEval subsets contain average trajectories of fewer than 200 words. In stark contrast, the Algorithmically-Generated and Hand-Crafted subsets of Who&When are significantly longer, averaging 1,500 and 7,460 words, respectively. While the average number of steps in ReasonEval is comparable to the Who&When Algorithmically-Generated subset, the exceptional length and intricacy of the Hand-Crafted subset establish it as the most demanding benchmark in our experiments.

Table 6: Most frequent faults in the Who&When dataset by generation method.

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

Table 7: 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

A.1 Limitations of Who&When Dataset

The Who&When dataset is one of the first open-source datasets to address the issue of fault attribution in agnetic systems. It advances the field by giving a new perspective on evaluating language-conditioned agentic systems. 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. We show the most frequent step-level and faults in Table 6 and Table 7, 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 Orchesterator. Epperson et al (Epperson et al., 2025) 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.

B Alternative Evaluation Metrics

To ensure clarity and standardize our evaluation, we adopt step-level accuracy as the primary, unified metric across all experiments. While prior work has proposed other metrics for specific datasets like Who&When (Zhang et al., 2025a) and ReasonEval (Xia et al., 2025), step-level accuracy provides the most consistent and stringent measure of performance for our purposes.

For instance, prior work on agentic trajectories has used an agent-level accuracy metric (Zhang et al., 2025a), which evaluates if the correct faulty agent is identified. However, our analysis of the Who&When dataset reveals a significant agent-level label imbalance: the WebSurfer agent is the root cause of failure in 57% of all trajectories. This skew allows a trivial baseline that always predicts this single agent to achieve 57% accuracy, rendering the metric less informative. As a more fine-grained measure, step-level accuracy is therefore our primary focus. For completeness, we provide agent-level comparisons in Appendix C.

Furthermore, the ReasonEval benchmark (Xia et al., 2025) employs step-level F1 and solution F1 scores. These metrics are necessary because the ReasonEval dataset contains both correct and incorrect trajectories, requiring models to first detect the presence of an error before localizing it and necessitating a separate solution-level metric. In contrast, Who&When only contains failed trajectories, simplifying the task to error localization. To maintain a clear and unified evaluation framework, we use step-level accuracy in the main text. Nevertheless, we report results using the F1 metrics in Appendix C to ensure direct comparison with the ReasonEval literature.

C Extended results on Who&When Dataset and ReasonEval Dataset using alternative metrics

In this section, we show results using alternative metrics and additional state-of-the-art methods to fully present the performance of RAFFLES. Notice that additional state-of-the-art methods have not been independently reproduced by the authors due to several factors, including computational resource limitations, lack of access to OpenAI closed-source models, or the absence of reproducible prompts in the original publications.

C.1 Who&When

On Who&When dataset, Table 8 shows both agent-level and step-level results with a diverse choice of model families of Llama 3.3 70B, Llama 3.1 8B, Mixtral 8x22B, gpt-oss-20b, and Claude Sonnet 4. Across these different model families, RAFFLES consistently demonstrates superior performance. It keeps the top rank for step-level accuracy in almost every model on both the algorithmically-generated and hand-crafted data splits. For agent-level accuracy, it consistently ranks either first or a close second. These results highlight the robustness and effectiveness regardless of the underlying language model.

When compared with other state-of-the-art methods (Table 9), RAFFLES establishes new performance benchmarks. Our method achieves the highest agent-level accuracy on both the algorithmically-generated (67.46%) and hand-crafted (67.24%) test sets, outperforming all prior work, including those that require simulation. Furthermore, RAFFLES also sets a new state-of-the-art in step-level accuracy on the algorithmically-generated data with a score of 51.59%.

Table 11 highlights a trade-off inherent to our methodology: the RAFFLES approach generally shows a higher average latency compared to the simple Chat-LLM baseline across all evaluated models and testing paradigms. This increased processing time is an expected consequence of our method's design, as its iterative, multi-step refinement process introduces additional inference calls compared to one-off LLM baselines.

C.2 ReasonEval

Although not primarily designed as a process reward model, RAFFLES demonstrates notable adaptability to other tasks, specifically fault attri-

Table 8: Full Results on the Who&When dataset, comparing RAFFLES against baseline methods across various models.

Model	Method	Algorithmically-generated		Hand-crafted	
		Step-level	Agent-level	Step-level	Agent-level
Llama 3.1 8B	**Chat-LLM**	15.08	**53.97**	6.90	**62.07**
	Routers - Step by Step	16.67	46.03	0.00	29.31
	Routers - Binary Search	7.14	23.81	5.17	50.00
	Tool-Caller	19.05	45.24	18.97	53.45
	RAFFLES	**34.92**	50.00	**25.86**	58.62
Llama 3.3 70B	**Chat-LLM**	19.05	55.56	6.90	**63.79**
	Routers - Step by Step	6.35	23.02	3.45	17.24
	Routers - Binary Search	4.76	32.54	10.34	53.45
	Tool-Caller	33.33	46.03	13.56	50.85
	RAFFLES	**43.65**	**61.90**	**20.69**	60.34
Mixtral 8x22B	**Chat-LLM**	20.63	35.71	1.75	52.63
	Routers - Step by Step	10.32	28.57	0.00	3.45
	Routers - Binary Search	0.79	33.33	10.34	**53.45**
	Tool-Caller	17.46	31.75	7.02	38.60
	RAFFLES	**29.37**	**47.62**	**15.79**	38.60
gpt-oss-20b	**Chat-LLM**	14.29	**63.49**	3.45	25.86
	Routers - Step by Step	19.84	34.92	22.41	58.62
	Routers - Binary Search	15.08	38.89	15.52	**53.45**
	Tool-Caller	29.37	41.27	17.24	39.66
	RAFFLES	**44.44**	58.73	**22.41**	53.45
Claude Sonnet 4	**Chat-LLM**	29.37	62.70	3.45	**60.34**
	Routers - Step by Step	30.16	38.89	12.07	27.59
	Routers - Binary Search	33.33	52.38	20.69	51.72
	Tool-Caller	30.95	41.27	18.97	51.72
	RAFFLES	**51.59**	**67.46**	**22.41**	**60.34**

Table 9: State-of-the-art agent-level and step-level results on the Who&When dataset, as reported in recent literature. These results have not been independently reproduced by the authors. An entry of 'N/A' in the 'Model' column signifies that the method does not require the use of large language models. 'Agent' represents agent-level accuracy and 'Step' represents step-level accuracy. Bold text shows the best performance in each category.

Category	Methods	Model	Algorithmically-generated		Hand-crafted	
			Agent	**Step**	**Agent**	**Step**
Baseline	All-at-once (Zhang et al., 2025a)	GPT-4o	51.12	13.53	53.44	3.51
	Step-by-step (Zhang et al., 2025a)	GPT-4o	26.02	15.31	32.75	8.77
	Binary Search (Zhang et al., 2025a)	GPT-4o	30.11	16.59	36.21	6.90
Requires Simulation	FAMAS (Ge et al., 2025)	Qwen2.5-72B	55.56	23.81	62.07	**41.38**
No Simulation	CORRECT (Yu et al., 2025)	GPT-5	–	38.10	–	17.20
	Causal Framework (Ma et al., 2025)	N/A	48.50	36.20	56.80	18.20
	RAFFLES (Ours)	Claude Sonnet 4	**67.46**	**51.59**	**67.24**	27.59

bution in reasoning chains. As detailed in Table 10, RAFFLES outperforms current state-of-the-art reported results on the ReasonEval dataset from recent literature. Our model achieves superior performance against both fine-tuned baselines, such as Llemma-34B, and models without task-specific fine-tuning, such as GPT-4. These results indicate that the RAFFLES framework can be effectively extended to fault attribution for reasoning chains.

Table 12: Percentage of changed candidate step t between current and prior iterations

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

D Extended Analysis of Iterative Reasoning

Figure 4 illustrates the 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 datasets converge toward the ground truth distribution. Focusing on the Hand-Crafted dataset in Figure 4b, 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 distribution converges towards that of the ground truth, showcasing the efficacy of the iterative process.

Similarly, in the ReasonEval GSM8K subset in Table 13, the benefit of iterative reasoning is more apparent. There is a clear improvement of accuracy and F1 score as iteration increases from $K = 0$ (only structured reasoning) to $K = 2$ (maximum 2 iterations) on all of the subsets. This presents strong evidence that iterative reasoning works.

On the Who&When dataset, an intriguing divergence of performance emerges in our experiments, as illustrated for the Llama 3.3 70B model in Figure 3. 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

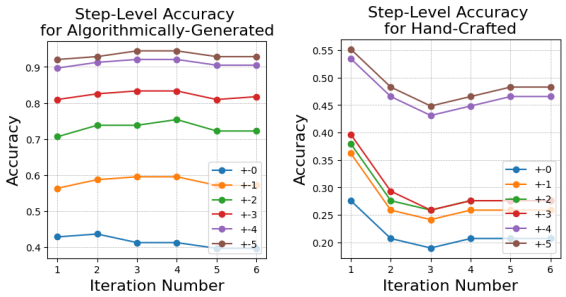


Figure 3: Llama 3.3 70B step-level accuracy vs. number of reasoning iterations

convergence.

The primary distinction between the Algorithmically-Generated (Figure 4a) and Hand-Crafted (Figure 4b) 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 4b. We hypothesize this is analogous to the "lost in the middle" phenomenon reported for LLMs in long-context scenarios (Liu et al., 2023a). 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.

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 3. Furthermore, we analyze the rate of convergence by computing the percentage of changed steps between consecutive iterations, presented in Table 12. 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 (Madaan et al., 2023), and highlights the necessity of an early stopping mechanism, such as a maximum iteration count K, to cap the reasoning

Table 10: State-of-the-art Solution F1 and Step F1 results on the ReasonEval dataset, as reported in recent literature. These results have not been independently reproduced by the authors. Bold values indicate the best performance in each column.

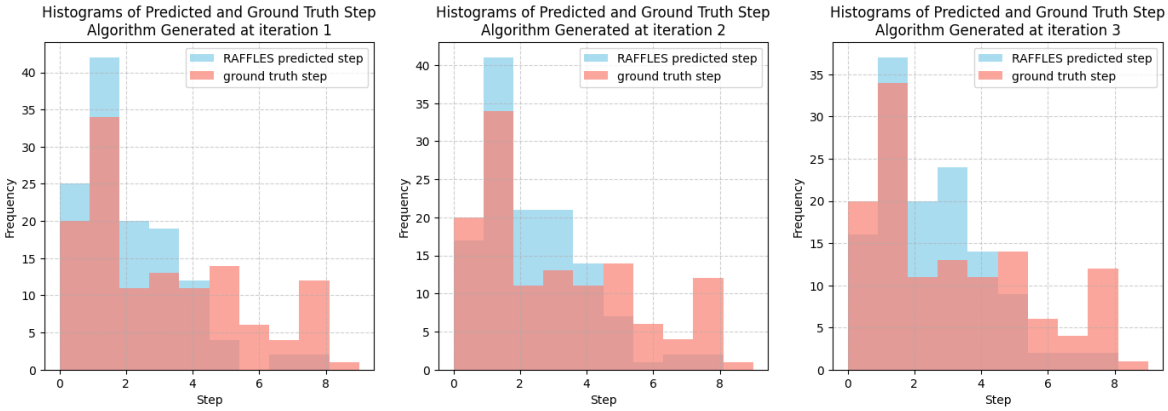
	MR-MATH-Invalid		MR-GSM8K-Original		MR-GSM8K-Reversed	
Method	**Solution F1**	**Step F1**	**Solution F1**	**Step F1**	**Solution F1**	**Step F1**
Previous SOTA results, as reported in Xia et al. (2025)						
GPT-4	73.2	61.0	81.7	69.0	72.2	52.2
Math-shepherd-Mistral-7b	70.1	60.0	86.0	73.4	77.2	59.6
ReasonEval WizardMath-7B-V1.1	78.6	73.9	74.1	72.8	74.4	70.5
ReasonEval Llemma-34B	79.6	77.5	81.0	73.5	76.1	69.3
Our experiments						
RAFFLES Llama 3.3 70B	84.3	75.7	89.8	76.9	84.2	65.2
RAFFLES Mixtral 8×22B	73.3	64.2	72.2	59.7	71.0	54.1
RAFFLES Claude Sonnet 4	**92.4**	**86.7**	95.1	82.3	91.7	70.7
RAFFLES gpt-oss-20b	89.3	79.8	**95.5**	**84.2**	**93.7**	**73.1**
Compared to previous best	+13.6%	+3.0%	+11.0%	+14.6%	+21.4%	+3.7%

Table 11: Average Latency (s) on Who&When datasets

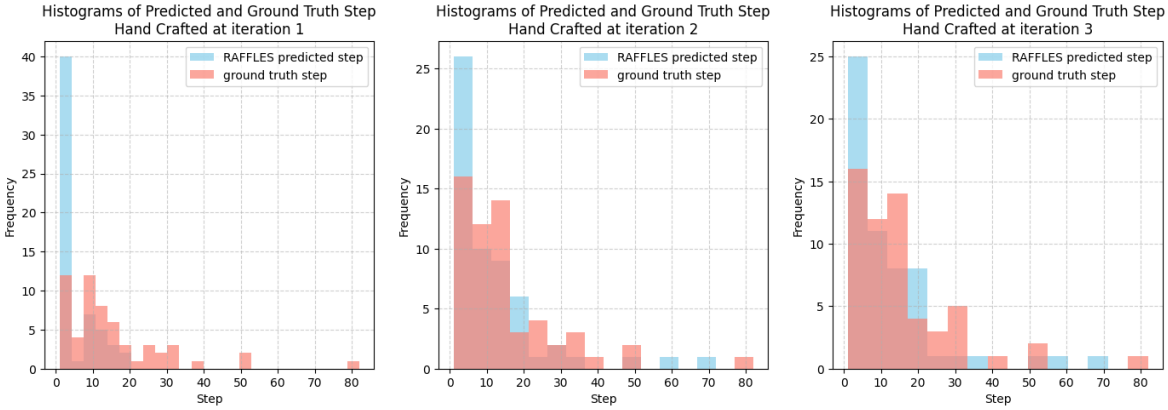
	Llama 3.3 70B		gpt-oss-20b		Claude Sonnet 4	
Method	**Alg.**	**Hand.**	**Alg.**	**Hand.**	**Alg.**	**Hand.**
Chat-LLM	8.69	7.81	10.22	34.80	5.94	8.49
Routers (Step by Step)	18.70	129.3	3.56	12.72	24.5	200.34
Routers (Binary Search)	20.16	32.98	3.54	7.89	10.39	21.15
Tool-Caller	32.36	53.40	12.69	24.73	24.86	40.79
RAFFLES	64.33	48.42	37.47	49.88	38.97	64.18

Table 13: Accuracy and F1 results for ReasonEval GSM8K subsets using Claude Sonnect 4 model

		Solution Acc.	Step Acc.	Solution F1	Step F1
MR-GSM8K-Original	judge	87.80	75.46	87.80	72.38
	K=0	94.36	82.65	94.34	79.34
	K=1	**95.13**	83.00	**95.13**	81.58
	K=2	95.06	**83.78**	95.06	**82.29**
MR-GSM8K-Reversed	judge	85.65	67.77	85.35	61.73
	K=0	87.86	70.42	87.46	64.39
	K=1	91.61	74.83	91.49	70.20
	K=2	**91.83**	**75.28**	**91.71**	**70.70**
MR-GSM8K-POT	judge	81.08	64.86	80.63	67.12
	K=0	86.94	72.97	86.82	74.67
	K=1	94.14	80.18	94.14	83.84
	K=2	**95.95**	**83.33**	**95.94**	**86.20**
MR-GSM8K-All	judge	86.33	71.19	86.25	66.07
	K=0	90.86	76.39	90.74	70.68
	K=1	93.46	79.09	93.44	75.56
	K=2	**93.66**	**79.89**	**93.64**	**76.28**



(a) Histogram of predicted and ground truth step for Algorithmically-Generated dataset, from iteration 1 to 3.



(b) Histogram of predicted and ground truth step for Hand-Crafted dataset, from iteration 1 to 3.

Figure 4: 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.

process.

E Prompt

E.1 Baseline prompt

We took the one-shot prompt in (Zhang et al., 2025a) 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 (Zhang et al., 2025a).

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 the location of the most critical mistake within
the provided segment. Determine which half of the segment contains the single step
where this crucial error occurs, ultimately leading to the failure in resolving
the user's query.
```

```
The problem to address is as follows: {problem}
Review the following conversation segment {range_description}:

{chat_segment_content}

Based on your analysis, predict whether the most critical error is more likely to
be located in the upper half ({upper_half_desc}) or the lower half
({lower_half_desc}) of this segment.
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": ""
}
```
```

Binary Search task output

```
Please answer in the format:
```json
{
    "judgement": "upper half" or "lower half",
    "reason": ""
}
```
```

E.2 Tool-Caller prompt

Our Tool-Caller mechanism is implemented using LangChain [1] 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.

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.
```

[1] https://www.langchain.com/

```
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. Our definition of faults is generic in the prompt, so that it works for different datasets, with only minor revisions needed to add a new task. When using the RAFFLES prompt on reasoning traces, we replace mentions of "agents" with "reasoning chains" and keep the mention of "steps".

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."
}}
```
```

F Example of RAFFLES reasoning process

An example RAFFLES reasoning process can be found in Figure 5 and Figure 6

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.

RAFFLES - Decisive Fault Attribution

Iteration

1

Judge
{
"agent_name": "Data_Extraction_Expert",
"step_number": "0",
}

Evaluator
Does the step and agent name match the log?
✅, confidence 100
Did the agent make a mistake? ❌, confidence 0
Is it the first mistake? ❌, confidence 0
Is mistake corrected? ✅, confidence 90

Evaluators confidence < 350, bad answer, judge again

Evaluator's Reasoning are fed to the judge

2

Judge
{
"agent_name": "Verification_Expert",
"step_number": "1",
}

Evaluator
Does the step and agent name match the log?
✅, confidence 100
Did the agent make a mistake? ✅, confidence 90
Is it the first mistake? ❓, confidence 70
Is mistake corrected? ✅, confidence 90

Evaluators confidence >= 350, good answer

Verification_Expert at step 1 is the decisive fault!

Figure 5: Example log from Who&When dataset, and how the 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 6: Continued example of the RAFFLES 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.