TRAIL: Trace Reasoning and Agentic Issue Localization

Anonymous ACL submission

Abstract

The increasing adoption of agentic workflows across diverse domains brings a critical need to scalably and systematically evaluate the complex traces these systems generate. Current evaluation methods depend on manual, domain-specific human analysis of lengthy workflow traces—an approach that does not scale with the growing complexity and volume of agentic outputs. Error analysis in these settings is further complicated by the interplay of external tool outputs and language model reasoning, making it more challenging than traditional software debugging. In this work, we (1) articulate the need for robust and dynamic evaluation methods for agentic workflow traces, (2) introduce a formal taxonomy of error types encountered in agentic systems, and (3) present a set of 148 large human-annotated traces (`TRAIL`) constructed using this taxonomy and grounded in established agentic benchmarks. To ensure ecological validity, we curate traces from both single and multi-agent systems, focusing on real-world applications such as software engineering and open-world information retrieval. Our evaluations reveal that modern long context LLMs perform poorly at trace debugging, with the best GEMINI-2.5-PRO model scoring a mere 11% on `TRAIL`. Our dataset and code are made publicly available to support and accelerate future research in scalable evaluation for agentic workflows[1].

1 Introduction

The rapid advancement of large language models (LLMs) has catalyzed the development of agentic systems capable of automating difficult, multi-step tasks across various domains such as software engineering and multi-hop IR (Ma et al., 2023; OpenAI, 2024; Nguyen et al., 2024; Wang et al., 2025a). Unlike traditional generative models, agents can interact with diverse tools and dynamically navigate environments, often with minimal human supervision (Wang et al., 2024a). This escalation of system complexity demands more challenging and multi-faceted evaluation processes (Nasim, 2025) and has led to the adoption of LLMs as evaluators for such agentic systems (Zheng et al., 2023; Chen et al., 2024; Kim et al., 2024; Zhu et al., 2025; Deshpande et al., 2024a).

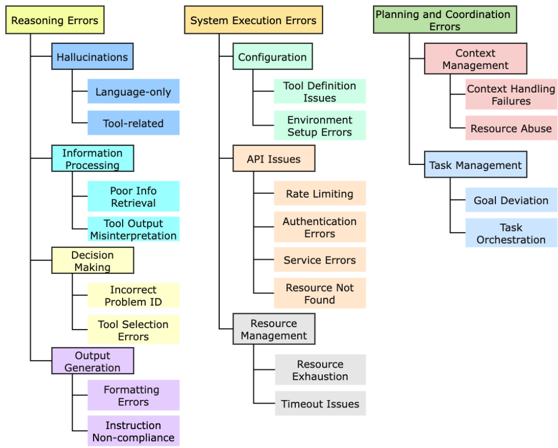


Figure 1: Illustration of the TRAIL taxonomy of errors

However, as multi-agent systems scale and become integral to real-world workflows, evaluating and debugging their performance remains a significant challenge. Agentic non-determinism (Laban et al., 2025; Patronus AI, 2025) and multi-step task solving (Mialon et al., 2023; Yao et al., 2024) demand greater observability than the simple end-to-end evaluations offered by existing benchmarks (Kapoor et al., 2024a; Zhuge et al., 2024; Moshkovich et al., 2025; Cemri et al., 2025). Such complex environments require granular taxonomies and well-annotated traces that can serve as references for debugging and root-cause analysis of agent behaviors (Cemri et al., 2025). When creating taxonomies and benchmarks to test and improve agents, we must ensure these are grounded in real-world applications and are not centered around

[1] Hidden for double-blind review

dummy data (Bowman and Dahl, 2021; Liu et al., 2024b). Previous agent trace analysis frameworks have primarily focused on parsed traces containing unstructured text (Cemri et al., 2025), which do not adequately represent common agent framework outputs that generate structured traces logged in standardized formats like `opentelemetry` (OpenTelemetry, 2025). Additionally, as observed by Guo et al. (2023); Sui et al. (2024), handling structured data remains challenging for LLMs, an observation corroborated by previous research on automated software engineering trace analysis (Roy et al., 2024a; Ma et al., 2024b). These limitations highlight the need for new approaches specifically designed for structured agentic traces. To address these challenges and facilitate the analysis and evaluation of agentic executions, we propose a formal error taxonomy, shown in Figure 3, that promotes granular failure diagnosis. We also present a carefully curated, turn-level annotated trace dataset called TRAIL (Trace Reasoning and Agentic Issue Localization), which demonstrates the validity and practical utility of our proposed taxonomy.

In our work, we utilize and build on SWE-Bench (Jimenez et al., 2024; Aleithan et al., 2024) and GAIA (Mialon et al., 2023) while addressing three major shortcomings inherent to previous automatic agent evaluation paradigms. Firstly, we aim to replace end to end analysis of agents with a benchmark containing step-level analysis of traced agentic workflows. Secondly, we address the need for grounding in real scenarios by producing `opentelemetry`-based structured traces that span beyond present model context length limits. Finally, as compared to benchmarks focused only on agentic reasoning and coordination (Cemri et al., 2025; Kokel et al., 2025), TRAIL focuses on validity through addition of finer, more aligned system execution failures and planning error categories such as *API errors* and *Task Orchestration Errors* to our taxonomy. Such categories are not only relevant to model developers but also to users and engineers optimizing single and multi-agent AI applications. The contributions of our work are as follows:

- We introduce a formal taxonomy (Figure 1) that defines, fine-grained agentic error categories spanning across three key areas: reasoning, planning, and execution.
- Based on this taxonomy, we present TRAIL, an ecologically grounded execution trace benchmark comprising 148 meticulously curated traces (totaling 1987 open telemetry spans, of which 575 exhibit at least one error) drawn from the GAIA (Mialon et al., 2023) and SWE-Bench (Jimenez et al., 2024) datasets and covering a wide range of tasks.
- We show that TRAIL is a non-trivially difficult benchmark for LLMs on many fronts
    1. Current SOTA LLM families such as O3, CLAUDE-3.7-SONNET and GEMINI-2.5-PRO perform modestly at best on TRAIL, both in terms of predicting error categories and their location. With GEMINI-2.5-PRO the best performing model, achieving only 11% combined joint accuracy on both splits.
    2. Solving TRAIL requires a significant fraction of the maximum input length of LLMs (or exceeds it), as well as requires generating significant fraction of their maximum output (See Table 2, Figure 5)
    3. Models benchmarked on TRAIL benefit from both the presence and greater extent of reasoning chains (§5.1.4, §5.1.5), highlighting the need for improvement in exploration capabilities of LLMs.
- TRAIL is fully open-source (MIT License), will be accompanied by a HuggingFace leaderboard, and serves as a foundation for future research on evaluating agentic workflows.

2 Relevant Work

LLM-as-a-Judge Shortcomings of conventional metrics such as ROUGE, BLEU, and BERTScore (Schluter, 2017; Freitag et al., 2020; Hanna and Bojar, 2021) has led to the wide adoption of LLMs as evaluators and critics of other AI systems (Zheng et al., 2023; Zhu et al., 2025; Chen et al., 2025, 2024; Kim et al., 2024). Recent approaches have enhanced LLM judges' reasoning capabilities through techniques like unconstrained evaluation plan and specialized training methods that enable more robust evaluation performance across diverse scenarios (Lightman et al., 2023; Wang et al., 2024e; Trivedi et al., 2024; Saha et al., 2025). The evaluation landscape has evolved significantly with the introduction of frameworks like FLASK (Ye et al., 2024b) which decompose coarse-level scoring into skill set-level evaluations for each instruction, demonstrating

```
"timestamp": "2025-03-24T15:05:02.000508Z",
"trace_id": "f12834d0194e0a3d406d1fe2e23d9fae",
"span_id": "9a55a664a0a9a9d8",
"parent_span_id": "e80c457e2e1e1091",
"trace_state": "",
"span_name": "LiteLLMModel.__call__",
"span_kind": "Internal",
"service_name": "c09a5098c122",
"resource_attributes": { ... },
"scope_name": "openinference.instrumentation.smolagents",
"scope_version": "0.1.8",
"span_attributes": {
    "input.mime_type": "application/json",
    "input.value": "{\"messages\": [{\"role\": \"system\", \"content\":
        [{\"type\": \"text\", \"text\": \"You are an expert assistant who
        can solve any task using code blobs. You will be given a task to
        solve as best you can.\\nTo do so, you have been given access to
        a list of tools: these tools are basically Python functions which
        you can call with code.\\nTo solve the task, you must plan
        forward to proceed in a series of steps, in a cycle of 'Thought:',
        'Code:', and 'Observation:' sequences. }",
    "llm.input_messages.0.message.content": ...
    "llm.input_messages.0.message.role": "system",
    ...
    "llm.input_messages.4.message.role": "tool-response",
    "llm.invocation_parameters": "{}",
    "llm.model_name": "anthropic/claude-3-7-sonnet-latest",
    ...
    "llm.token_count.completion": "259",
    "llm.token_count.prompt": "5131",
    "llm.token_count.total": "5390",
    "openinference.span.kind": "LLM",
    "output.mime_type": "application/json",
    "output.value": ...,
    "pat.app": "SWEBench",
    "pat.project.id": "882e0ea9-9076-4806-918b-4a143037a1f1",
    "pat.project.name": "swe-bench-dev"
    }
```

```
"category": "Tool Selection Error",
"location": "e399aa27e024a138",
"evidence": "task = (...)\\nprint(task)"
"description": "The agent's thought said: 'I'll now call search_agent
    with this detailed task.' However, the 'Code:' generated printed the
    task through the interpreter instead of calling agent",
"impact": "MEDIUM"
```

```
"category": "Resource Abuse",
"location": "9a55a664a0a9a9d8",
"evidence": "Code:\n``parser_files = [file for file in tree if 'parser' in
    file.lower() and file.endswith('.py')]\nprint(\"Parser-related files (first
    20):\")\nfor i, file in enumerate(parser_files[:20]):\nprint(file)\n\n...````",
"description": "There is a problem with the way it wants to extract and print
    the tree, as it will not print the lines line by line.",
"impact": "MEDIUM"
```

```
"category": "Instruction Non-compliance",
"location": "61c56440907bf40a",
"evidence": "{'input.mime_type': 'application/json', 'input.value':
        '{\"messages\": [{\"role\": \"system\", \"content\": [{\"type\":
        \"text\", \"text\":..."},
"description": "The model didn't submit the final answer as a direct patch but
        instead provided info about repository",
"impact": "HIGH"
```

```
"category": "Context Handling Failures",
"location": "03d52712671e1730",
"evidence": "output = gitnigest("https://github/...")\nprint(output)"
"description": "The model prints the entire output when the system
        instructions specify to not print more than 500 characters",
"impact": "MEDIUM"
```

```
"category": "Formatting Errors",
"location": "9a55a664a0a9a9d8",
"evidence": "Tree structure (first 20
    entries):\nD\ni\nr\ne\nc\nt\no\nr\ny\n\ns\nt\nr\nu\nc\nt\nu\nr\ne\n:",
"description": "The model prints first 20 chars instead of printing repo tree",
"impact": "LOW"
```

```
"category": "Language-only",
"location": "e3ac5de23c0ba0e8",
"evidence": "Thought: The tree variable doesn't seem to contain file paths
    as I expected",
"description": "The model says \"The tree variable doesn't seem to contain
file paths as I expected\", without any evidence or additional exploration",
"impact": "HIGH"
```

```
"category": "Rate Limiting",
"location": "61c56440907bf40a",
"evidence": "status_message: \"RateLimitError: litellm.RateLimitError:
        AnthropicException - ...",
"description": "The API call to Anthropic is rate limited leading to failure..",
"impact": "HIGH"
```


Figure 2: TRAIL trace's span structure and error examples

high correlation between model-based and human-based evaluations. The Prometheus models (Kim et al., 2023, 2024, 2025) established a significant benchmark by creating judge models that surpass GPT-4 in ranking for subjective evaluation criteria. Their research also examined how performance deteriorates as subjectivity increases. More recently, several studies have enhanced judge model performance through external augmentations and checklists, highlighting the importance of incorporating high-quality reasoning chains and human guidance in model training (Lee et al., 2025; Deshpande et al., 2024b,a; Chen et al., 2025; Wang et al., 2025b). Despite promising advancements, LLM judges have shown issues with propagation of biases and lack of robustness to longer inputs (Ye et al., 2024a; Hu et al., 2024b; Wei et al., 2024; Zhou et al., 2025). Since trace evaluation requires robust reasoning over large contexts (Tian et al., 2024), LLM judges have not seen wide application in this sector yet.

Agentic Evaluation LLM-powered agents have gained significant traction for their capacity to manage intricate, sequential tasks while adaptively engaging with varied environments, rendering them particularly valuable for practical real-world applications such as software engineering and multi-hop IR (Ma et al., 2023; OpenAI, 2024; Nguyen et al., 2024; Wang et al., 2025a; Jimenez et al., 2024; Qian et al., 2024; Wang et al., 2024d; Patil et al., 2024). However, the performance gains of multi-agent frameworks remain minimal compared to their single-agent counterparts (Xia et al., 2024; Kapoor et al., 2024b). As these agentic systems become more prevalent, evaluation frameworks (as compared to LLM evaluation) must offer greater customization and granularity to effectively assess the complex and sometimes unpredictable interactions between multiple agents, enabling users to precisely identify and diagnose errors at each step of the process (Roy et al., 2024b; Akhtar et al., 2025; Jiang et al., 2025; Zhuge et al., 2024; Open-Manus, 2024).

Agent Benchmarks Software engineering domain has become a fertile testbed for LLM-based collaborative problem solving for real-world use cases and to evaluate agents' ability to handle realistic coding tasks. SWE-Bench (Jimenez et al., 2024; Aleithan et al., 2024; Pan et al., 2024) was introduced as a grounded benchmark asking whether LLMs can resolve real-world GitHub issues. Similarly, GAIA (Mialon et al., 2023) is a benchmark for General AI Assistants featuring real-world questions requiring reasoning, tool use, and multimodality. AssistantBench (Yoran et al., 2024) introduces a challenging benchmark of realistic, time-consuming web tasks to evaluate web agents. For agents, it is key to distinguish input sample failures from the judge model's own internal reasoning failures. Highlighting spans can help models focus and avoid losing context while also providing additional explainability and performance improvements (Lv et al., 2024; Li et al., 2024). Other core benchmarks include DevAI (Zhuge et al., 2024), MLE-bench (Chan et al., 2024), HumanEval (Du et al., 2024), and MBPP (Odena et al., 2021).

Traces and Error Taxonomies Emerging work has emphasized the need for better observability in the agent execution traces to diagnose and manage the non-deterministic nature of agentic systems (Kapoor et al., 2024a; Zhuge et al., 2024; Moshkovich et al., 2025; Cemri et al., 2025). For instance, Roy et al. (2024a) explores using LLM-based agents to dynamically collect diagnostic information from logs and metrics using retrieval tools for root cause analysis of cloud system incidents. Akhtar et al. (2025) surveys how LLMs are being applied to automate even log analysis in security contexts. Jiang et al. (2025) is a log analysis framework for diagnosing large-scale LLM failures based on studying real-world training failures. Ma et al. (2024c) explores the potential for log parsing by proposing an LLMParser delivering comprehensive evaluations in various settings. Once the trace errors are found, to serve as references for users to debug or conduct root cause analysis of agent behaviors, these errors require a granular taxonomy (Cemri et al., 2025; Kokel et al., 2025; Bai et al., 2024a). MAST (Cemri et al., 2025) presents an empirically grounded failure mode taxonomy but focusing only on agentic reasoning and coordination. ACPBench (Kokel et al., 2025), using a synthetic dataset, focuses on atomic reasoning about action and is designed to evaluate LLM's core planning skills. Other related work includes taxonomies to evaluate multi-turn conversations (Bai et al., 2024a) and designing LLM agent framework to identify and quantify complex evaluation criteria (Arabzadeh et al., 2024; Epperson et al., 2025).

Thus, TRAIL distinguishes itself through its ecological validity while comprehensively addressing both single and multi-turn systems with its granular taxonomy, particularly emphasizing critical execution and planning failure patterns.

3 Agentic Error Taxonomy

LLM reasoning, while having advanced significantly, remains a critical source of failures in agentic workflows (Costarelli et al., 2024). These errors span several dimensions, from flawed information generation to problematic decision-making and output production (Cemri et al., 2025). In this section, we define a comprehensive taxonomy (as summarized in Figure 3) of agentic errors spanning three key areas of failures: reasoning, planning and coordination, and system execution.

3.1 Reasoning Errors

Hallucinations LLMs frequently generate factually incorrect or nonsensical content, a problem that also affects agents (Huang et al., 2025; Ji et al., 2023). *Text-only* hallucinations include fabricated or ungrounded statements that conflict with real-world knowledge (Ji et al., 2023). In contrast, *Tool-related* hallucinations arise when agents invent tool outputs or misunderstand tool functions, such as fabricating results or claiming nonexistent capabilities (Zhang et al., 2024b; Xu et al., 2024).

Information Processing Retrieval-augmented generation, which retrieves and reasons over data relevant to a query, has become increasingly popular (Hu and Lu, 2024; Gao et al., 2025). However, recent work (Xu et al., 2025; Su et al., 2025) shows that LLMs and agents often struggle to reason effectively over retrieved information. These issues can be grouped into two main types: poor information retrieval and misinterpretation of outputs. *Poor information retrieval* (Wu et al., 2024) can introduce redundancy and content overload (Stechly et al., 2024), while misinterpretation of retrieved context (*Tool output Misinterpretation*) (Karpinska et al., 2024; Wang et al., 2024b) may cause errors that propagate throughout an agent's reasoning process, leading to broader incorrectness or inefficiencies.

Decision Making Task misunderstanding at the step level often arises from ambiguous prompts, unclear instructions, or an LLM's inability to distinguish between prompt and data instructions (Zverev et al., 2024). Detecting such misunderstandings (*Incorrect Problem ID*) requires analyzing an agent's path, which is challenging in large contexts (Yuan et al., 2024) and reliable detection of these errors is crucial for agent improvement. Furthermore, effective decision making in agent workflows also depends on selecting the appropriate tool at each step (Qin et al., 2023). Because optimal planning and tool selection reduces cost and increases efficiency (Yehudai et al., 2025), we place *Tool Selection Error* under *Decision Making*.

Output Generation LLMs often produce incorrectly formatted structured outputs (Shorten et al., 2024; Liu et al., 2024a), which is problematic for tool calls that need precise JSON or code formatting. To capture this, our taxonomy includes *Formatting Errors*. Similarly, LLMs frequently struggle following complex/ambiguous instructions (White et al., 2024; Heo et al., 2024), hence we subcategorize *Instruction Non-compliance*.

3.2 System Execution Errors

Configuration Issues Incorrect agentic environment configuration can cause failures and limit agent capability (Hu et al., 2024a). One key issue is *Incorrect Tool Definition*, as shown by Fu et al. (2024), agents can be misled by inaccurate or obfuscated tool definitions in prompts, posing security and reliability risks. Additionally, poor setup of environment variables (*Environment Setup Errors*), e.g., missing API keys or incorrect file permissions, can cause unexpected failures and disrupt reasoning paths.

API and System Issues As agentic systems combine LLMs with software tools, tool usage or implementation errors can disrupt workflows. With the rise of remote tool access via protocols like MCP (Anthropic, 2025), capturing and categorizing API failures is increasingly important for prompt reporting to tool developers (Shen, 2024). Runtime errors involving agentic tools remain underexplored (Milev et al., 2025), so we specifically include the most common API tool errors in our taxonomy: *Rate Limiting* (429), *Authentication Errors* (401, 403), *Service Errors* (500), and *Resource Not Found* (404) (Liu et al., 2023a).

Resource Management Resource management is crucial for agents using operating system tools like interpreters or terminals. Poor task planning can expose vulnerabilities, such as *Resource Exhaustion* from overallocation (Ge et al., 2023) or *Timeout Issues* from infinite loops (Zhang et al., 2024a), potentially causing memory overflows or system overloads. Early detection of these errors is vital to prevent infrastructure failures.

3.3 Planning and Coordination Errors

Context Management As planning and reasoning become integral to agentic workflows (Yao et al., 2023; Ke et al., 2025), agents must manage long-term context, including episodic and semantic information (Zhang et al., 2024c). In our taxonomy, we categorize failures in context or instruction retention as *Context Handling Failures*. Additionally, repeated tool calls (Kokane et al., 2024) (*Resource Abuse*) reflect shortcomings in planning, context management, and tool use, which our taxonomy also captures.

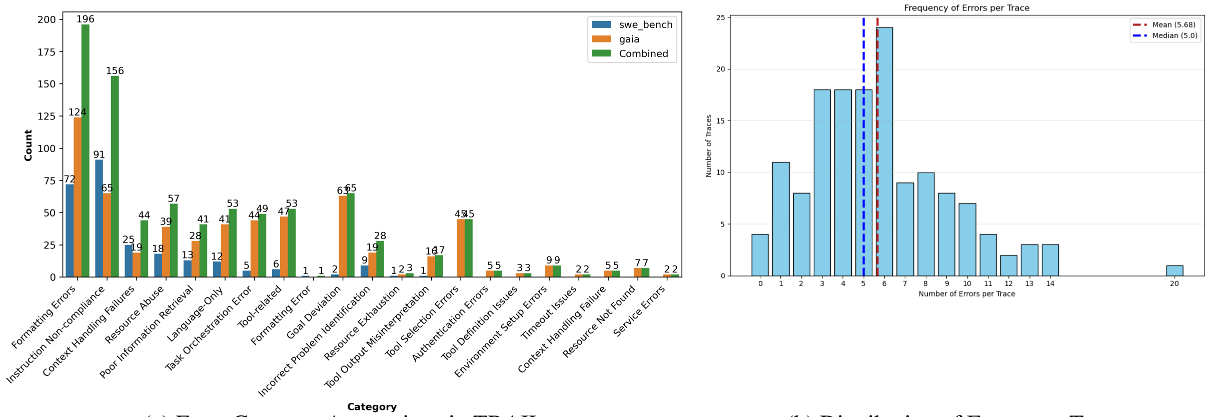
Task Management Environmental misconfigurations or LLM hallucinations can distract agentic systems, and poor recovery from such distractions often leads to goal deviation (Ma et al., 2024a). These issues are amplified in multi-agent setups with sub-tasks, making effective task orchestration crucial. Therefore, we include *Goal Deviation* and *Task Orchestration Errors* in our taxonomy.

4 TRAIL Benchmark

TRAIL is a benchmark aimed to evaluate LLM capabilities to analyze and evaluate long, structured, `opentelemetry` standardized agentic executions. TRAIL follows our fine grained taxonomy and contains 148 carefully annotated agentic traces. The dataset uses text-only data instances from the GAIA (Mialon et al., 2023) and SWE Bench Lite (Jimenez et al., 2024) datasets, spanning multiple information retrieval and software bug fixing tasks. It contains a total of 841 annotated errors, averaging at 5.68 errors per trace Figure 3.

4.1 Goals and Design Choices

Core Agent Task We aim to showcase realistic agentic workflows and so we target two widely adopted agentic datasets, the GAIA benchmark (Mialon et al., 2023), an open world search task, and the SWE-Bench-Lite (Jimenez et al., 2024) dataset, for locating and fixing issues in Github repositories. We select these datasets due to their challenging nature and necessity for environment and search space exploration.



(a) Error Category Annotations in TRAIL (b) Distribution of Errors per Trace

Figure 3: TRAIL Dataset Statistics

Agent Orchestration Liu et al. (2023b) first presented a standardized hierarchical method of orchestrating agents, derivatives of which are actively adopted by several works (Zhao et al., 2024, 2025). We closely follow this hierarchical structure and adopt the Hugging Face OpenDeepResearch agent (Hugging Face, 2024) for creating traces for the GAIA benchmark. We select the state-of-the-art `o3-mini-2025-01-31` (OpenAI, 2025d) and assign it as the backbone model for the manager and search agents respectively because of its strong tool use and planning ability as showcased by Phan et al. (2025). For more information, refer to §A.10.

Parallelly, to explore single-agent planning errors and elicit context handling errors for the SWE-Bench split, we use a CodeAct agent (Wang et al., 2024c) and provide it access to a sandboxed environment, a python interpreter and the `gitingest`[2] library. We select `claude-3-7-sonnet-20250219` as the backbone model due to its strong performance on software engineering tasks (Anthropic, 2025). To further organically introduce errors into this agent system, we add instructional constraints such as output length limits and force exploration via prompts. The complete prompt is at §A.12.

Workflow Tracing To ensure compatibility of this dataset with real world tracing and observability software, all traces are collected via `opentelemetry` (OpenTelemetry, 2025), specifically, its most widely adopted open-source derivative compatible with agents, the `openinference` standard (Arize AI, 2025) as adopted by Moshkovich et al. (2025).

[2]`https://github.com/cyclotruc/gitingest`

4.2 Data Annotation and Validation

We selected four annotators with expertise in software engineering and log debugging to label our agent traces. To assess agreement, a separate set of 63 traces was assigned. Results based on these indicate high inter-annotator agreement during curation. We defer details of our complete annotation and agreement measuring processes and actual numbers from them to §A.7.

4.3 Dataset Analysis

Following the post-annotation review, we found errors in 114 GAIA traces and 30 from SWE Bench. As shown in Figure 3, these errors cover various categories, with most falling under *Output Generation*. Specifically, *Formatting Errors* and *Instruction Non-compliance* make up 353 of 841 total errors—nearly 42%. In contrast, *System Execution Errors* are rare. This categorical imbalance highlights two important considerations for evaluating agentic pipelines. First, the prevalence of *Output Generation* errors suggests that current LLM systems struggle with high-level reasoning and understanding task parameters, even with careful prompt-engineering. Second, although infrequent, errors in categories like API failures can be catastrophic and are critical to detect, as they are often difficult to recover from, unlike errors due to goal deviation or tool misinterpretation. Most errors in our data are high or medium impact (Figure 6a). While model hallucinations and resource management issues greatly affect agent behavior, about 44% of *Output Generation* errors are low impact (Figure 6b). This underscores need for a classification scheme that includes rare but significant error types. A key feature of our taxonomy is ability to

Model	TRAIL (GAIA)				TRAIL (SWE Bench)			
	Cat. F1	Loc. Acc.	Joint	ρ	Cat. F1	Loc. Acc.	Joint	ρ
Llama-4-Scout-17B-16E-Instruct[†]	0.041	0.000	0.000	0.134	0.050	0.000	0.000	0.264
Llama-4-Maverick-17B-128E-Instruct[†]	0.122	0.023	0.000	0.338	0.191	0.083	0.000	-0.273
GPT-4.1[†]	0.218	0.107	0.028	0.411	0.166	0.000	0.000	0.153
Open AI o1[*]	0.138	0.040	0.013	0.450	CLE	CLE	CLE	CLE
Open AI o3[*]	0.296	0.535	0.092	0.449	CLE	CLE	CLE	CLE
Anthropic Claude-3.7-Sonnet[*]	0.254	0.204	0.047	**0.738**	CLE	CLE	CLE	CLE
Gemini-2.5-Pro-Preview-05-06[*†]	**0.389**	**0.546**	**0.183**	0.462	0.148	**0.238**	**0.050**	**0.817**
Gemini-2.5-Flash-Preview-04-17[*†]	0.337	0.372	0.100	0.550	**0.213**	0.060	0.000	0.292

Table 1: Performance across LLMs for Error Categorization & Localization on TRAIL (GAIA) and TRAIL (SWE Bench). Models marked with [*] have reasoning set to "high"; † indicates 1M+ token context window. Insufficient context length is marked as CLE. Pearson correlation b/w overall human and generated scores is shown under ρ.[3]

categorize well such long-tail, high-impact errors.

4.4 Summary of Evaluation Setup

For empirically evaluating and comparing LLM performance on TRAIL we choose the following LLMs — GPT-4.1, o1, o3, GEMINI-2.5 (both PRO+FLASH), CLAUDE-3.7-SONNET and LLAMA-4 (both Maverick+Scout). We defer detailed discussion of more evaluation setup specifics to A.3

5 Results

In §5.1, we analyze the research questions below:

- How does long context reasoning affect TRAIL performance? How many inputs exceed the LLM's context window? How does trace length impact this? We address these in §5.1.1 §5.1.2, and §5.1.3.
- Does TRAIL benefit from more reasoning? We explore this in §5.1.4 and §5.1.5.
- Which error categories are easier to predict? Where do non-reasoning models perform notably worse? We examine this in §5.1.6.

5.1 Qualitative and Quantitative Analysis

5.1.1 Task Difficulty - Context Length and Generation Horizon

As seen in Table 2, the distribution of raw JSON input token lengths injested to perform our task cuts close to the input context limit of several LLMs - with the maximum input trace length always being twice longer than the input length limit, and even the mean itself sometimes going over. Furthermore, even the typical output token length horizon the LLMs need to generate for the task exceeds the 1K tokens on average, with the maximum being ≈3.7K at the least. Besides being a significant % of the maximum output length, this indicates the difficultly long generation horizon TRAIL needs.

5.1.2 Long Context Ability and Model Performance

We compare how the models in Table 1 rank based on their aggregate performance on TRAIL vis-a-vis the relative ranking of the subsets of these models that occur on updated long-context benchmark leaderboards Longbenchv2 and fiction.live's LongContextBench (Bai et al., 2024b; Ficlive, 2025), and notice this differs for only one model (o3 being third best rather than best on the latter). We defer the complete detail of these rankings to §A.2.

5.1.3 Performance vs Input Length

We find all performance metrics to be anti-correlated with input length, as detailed in Table 3. This supports the hypothesis that longer input raw traces increase the difficulty of TRAIL for models.

5.1.4 Reasoning vs Non-Reasoning Models

From Table 1, we see all reasoning models except o1 outperforming non-reasoning ones on both Error category F1 and Location Accuracy. On Joint Accuracy, the gap between the two families is larger — Reasoning models other than o1 perform at 1.5-8 times the best performing non-reasoning model.

5.1.5 Does Reasoning Effort Matter?

To systematically assess the impact of reasoning extent, we experiment with the same model (o3) at "high," "medium," and "low" reasoning effort levels, as set by OpenAI's *reasoning.effort* parameter. We find that all three metrics, including Category F1 (0.296 → 0.277 → 0.264), decrease as reasoning effort decreases. These results empirically sup-

[3] All reported results are an average of three runs.

Task	Tokenizer	Input Limit	Output Limit	Input Context Lengths				Output Token Lengths			
				Min	Max	Mean	StdDev	Min	Max	Mean	StdDev
GAIA	gpt-4.1 (=o3)	1M	32.77K	20.94K	**7.50M**	286.85K	768.85K	0.11K	4.47K	1.11K	0.69K
GAIA	gemini-2.5	1M	8.19K	23.09K	**8.25M**	313.49K	843.53K	0.13K	4.95K	1.20K	0.75K
GAIA	claude-3.7	200K	128K	23.67K	**2.66M**	**262.67K**	456.64K	0.12K	5.37K	1.23K	0.78K
SWEBench	gpt-4.1 (=o3)	1M	32.77K	120.40K	**2.05M**	616.92K	473.05K	0.11K	3.71K	1.71K	0.75K
SWEBench	gemini-2.5	1M	8.19K	134.88K	**2.21M**	698.09K	552.34K	0.13K	4.09K	1.88K	0.83K
SWEBench	claude-3.7	200K	128K	140.16K	**2.43M**	**727.75K**	557.86K	0.12K	4.17K	1.93K	0.87K

Table 2: Input Context Lengths and Human-Annotated Output Token Lengths Across both GAIA and SWEBench Tasks and various SOTA models and their tokenizers. Input Length aggregates that exceed the limit are **highlighted**.

Corr.	**Location Acc**	**Joint Acc**	**Categ. F1**
Pearson (r)	-0.379	-0.291	-0.296
Spearman (ρ)	-0.508	-0.349	-0.225

Table 3: Correlations b/w Input Length & Performance

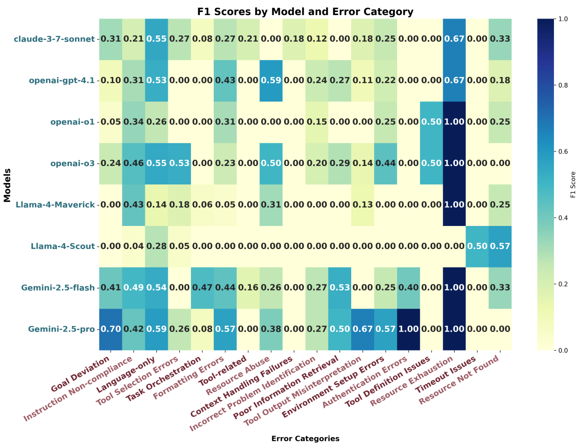


Figure 4: Heatmap for Error Category F1 across models; categories are ordered left to right based on their support

port that TRAIL performance benefits from higher reasoning effort at test time, and that the superior results for reasoning models are not solely due to improved pre- or post-training (§5.1.4). Full ablation results are in Appendix §A.4.

5.1.6 Performance Across Categories

Hard-to-Predict Categories Among the most challenging categories, *Context Handling Failures* stand out, as nearly all models score an F1 of 0.00, indicating these errors demand advanced reasoning. The only exception is CLAUDE-3.7-SONNET, which achieves a relatively better score of 0.18. *Tool Selection Errors* are also difficult to predict, with most models scoring between 0.00 and 0.08, apart from GEMINI-2.5-PRO (0.26), CLAUDE-3.7-SONNET (0.27), and especially O3 (0.53), suggesting this is a complex error type. Similarly, *Task Orchestration* shows uniformly low scores across models (0.00–0.08) except for GEMINI-2.5-FLASH, which stands out with a much higher F1 of 0.47.

Interesting Performance Divergence There are also categories where model performance diverges interestingly. For *Goal Deviation*, GEMINI-2.5-PRO and GEMINI-2.5-FLASH perform best (0.70 and 0.41, respectively), while CLAUDE-3.7-SONNET and O3 perform moderately (0.31, 0.24); O1 and other non-reasoning models score the lowest (≤ 0.05). In the case of *Poor Information Retrieval*, the two Gemini models are again notably better (0.50 and 0.53), with others at <0.30, suggesting better diagnosis of failures related to context.

Other Surprising Patterns *Language-Only* errors, a subtype of hallucination, are detected relatively well by all models (0.14–0.59), implying that these are easier for models to predict even without advanced reasoning capabilities. For *Formatting Errors*, performance is non-monotonic: GPT-4.1 (0.43) and the GEMINI-2.5 models (0.44–0.57) perform well, while O1, O3, and CLAUDE-3.7-SONNET perform worse (0.23–0.31). It is notable that O1 and GPT-4.1 outscore O3 on this category, despite being older and non-reasoning respectively. We defer some model-specific observations to §A.6

6 Conclusion

In this work, TRAIL, a new taxonomy for classifying agentic errors, along with an expert-curated dataset of 148 agentic problem instances and 841 unique errors from GAIA and SWE Bench. Current SOTA models perform poorly as LLM Judges on this dataset, with GEMINI 2.5-PRO achieving only 18% joint accuracy on GAIA and 5% on SWE Bench; three out of eight models cannot even process the full context. These results highlight that existing models struggle to systematically evaluate complex agentic traces, due to the inherent complexity of agentic systems and LLM context limitations. A new framework is needed for scalable, systematic evaluation of agentic workflows.

Limitations

The TRAIL dataset and taxonomy are primarily focused on text-only inputs and outputs but recent advancements in multimodal agentic systems require careful extension of the taxonomy to handle errors arising from new categories such as multimodal tool use. One additional limitation of TRAIL is the large number of tail categories with very few examples. It is important to ensure correctness of LLM-Judges on these categories due to the high-impact nature of the failures. Future research work can look into synthetic data generation for high-impact, low-occurrence categories by systematically modifying existing traces to induce catastrophic irrecoverable failures within the LLM context.

Ethics Statement

While curating this dataset, we ensure that annotators are only selected based on their age (18+) and their expertise in the computer science field. Annotator selection was not based on nationality, language, gender or any other characteristic apart from these two criteria. We pay annotators a total of $12.66 per trace where each trace takes 30-40 minutes to annotate. We ensure that the traces do not contain any PII or any explicit or biased content by manually verifying traces before forwarding these to annotators. The annotators were made aware of the open-sourcing of their work and consent was obtained beforehand.

Acknowledgments

We would like to acknowledge industry AI practicioners: Sam Yang, Mark Klein, Pasha Rayan and Pennie Li for their feedback on our error taxonomy.

References

Siraaj Akhtar, Saad Khan, and Simon Parkinson. 2025. Llm-based event log analysis techniques: A survey. *arXiv preprint arXiv:2502.00677*.

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. 2024. Swe-bench+: Enhanced coding benchmark for llms. *arXiv preprint arXiv:2410.06992*.

Anthropic. 2025. Claude 3.7 sonnet. https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: May 9, 2025.

Anthropic. 2025. Model context protocol: Transparency and control for ai inputs and outputs. Accessed: 2025-05-08.

Negar Arabzadeh, Siqing Huo, Nikhil Mehta, Qingyun Wu, Chi Wang, Ahmed Hassan Awadallah, Charles L. A. Clarke, and Julia Kiseleva. 2024. Assessing and verifying task utility in LLM-powered applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21868–21888, Miami, Florida, USA. Association for Computational Linguistics.

Arize AI. 2025. Openinference. https://github.com/Arize-ai/openinference. Accessed: May 9, 2025.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024a. MT-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7421–7454, Bangkok, Thailand. Association for Computational Linguistics.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. 2024b. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*.

Samuel R Bowman and George E Dahl. 2021. What will it take to fix benchmarking in natural language understanding? *arXiv preprint arXiv:2104.02145*.

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

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal A. Patwardhan, Lilian Weng, and Aleksander Mkadry. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *ArXiv*, abs/2410.07095.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. 2024. Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Nuo Chen, Zhiyuan Hu, Qingyun Zou, Jiaying Wu, Qian Wang, Bryan Hooi, and Bingsheng He. 2025. Judgelrm: Large reasoning models as a judge.

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. 2024. Gamebench: Evaluating strategic reasoning abilities of llm agents. *arXiv preprint arXiv:2406.06613*.

Google DeepMind. 2025. Gemini model thinking updates: March 2025. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. Accessed: May 11, 2025.

Darshan Deshpande, Selvan Sunitha Ravi, Sky CH Wang, Bartosz Mielczarek, Anand Kannappan, and Rebecca Qian. 2024a. Glider: Grading llm interactions and decisions using explainable ranking. *arXiv preprint arXiv:2412.14140*.

Darshan Deshpande, Zhivar Sourati, Filip Ilievski, and Fred Morstatter. 2024b. Contextualizing argument quality assessment with relevant knowledge. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 316–326, Mexico City, Mexico. Association for Computational Linguistics.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Will Epperson, Gagan Bansal, Victor Dibia, Adam Fourney, Jack Gerrits, Erkang Zhu, and Saleema Amershi. 2025. Interactive debugging and steering of multi-agent ai systems. *arXiv preprint arXiv:2503.02068*.

Ficlive. 2025. Fiction.livebench (april 6, 2025). https://fiction.live/stories/Fiction-livebench-April6-2025/oQdzQvKHw8JyXbN87. Accessed: 2025-05-12.

Markus Freitag, David Grangier, and Isaac Caswell. 2020. BLEU might be guilty but references are not innocent. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 61–71, Online. Association for Computational Linguistics.

Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2024. Imprompter: Tricking llm agents into improper tool use. *arXiv preprint arXiv:2410.14923*.

Yunfan Gao, Yun Xiong, Yijie Zhong, Yuxi Bi, Ming Xue, and Haofen Wang. 2025. Synergizing rag and reasoning: A systematic review.

Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023. Llm as os, agents as apps: Envisioning aios, agents and the aios-agent ecosystem. *arXiv preprint arXiv:2312.03815*.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Michael Hanna and Ondřej Bojar. 2021. A fine-grained analysis of BERTScore. In *Proceedings of the Sixth Conference on Machine Translation*, pages 507–517, Online. Association for Computational Linguistics.

Juyeon Heo, Miao Xiong, Christina Heinze-Deml, and Jaya Narain. 2024. Do llms estimate uncertainty well in instruction-following? *arXiv preprint arXiv:2410.14582*.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. 2024a. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. *arXiv preprint arXiv:2408.00764*.

Yucheng Hu and Yuxing Lu. 2024. Rag and rau: A survey on retrieval-augmented language model in natural language processing.

Zhengyu Hu, Linxin Song, Jieyu Zhang, Zheyuan Xiao, Jingang Wang, Zhenyu Chen, Jieyu Zhao, and Hui Xiong. 2024b. Rethinking llm-based preference evaluation. *arXiv e-prints*, pages arXiv–2407.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2).

Hugging Face. 2024. open deep research: An open-source replication of openai's deep research agent. https://github.com/huggingface/smolagents/tree/main/examples/open_deep_research.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

Zhihan Jiang, Junjie Huang, Zhuangbin Chen, Yichen Li, Guangba Yu, Cong Feng, Yongqiang Yang, Zengyin Yang, and Michael R. Lyu. 2025. L4: Diagnosing large-scale llm training failures via automated log analysis. *arXiv preprint arXiv:2503.20263*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024a. Ai agents that matter. *arXiv preprint arXiv:2407.01502*.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024b. Ai agents that matter. *arXiv preprint arXiv:2407.01502*.

Marzena Karpinska, Katherine Thai, Kyle Lo, Tanya Goyal, and Mohit Iyyer. 2024. One thousand and one pairs: A" novel" challenge for long-context language models. *arXiv preprint arXiv:2406.16264*.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, et al. 2025. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2023. Prometheus: Inducing fine-grained evaluation capability in language models. *arXiv preprint arXiv:2310.08491*.

Seungone Kim, Juyoung Suk, Ji Yong Cho, Shayne Longpre, Chaeeun Kim, Dongkeun Yoon, Guijin Son, Yejin Cho, Sheikh Shafayat, Jinheon Baek, Sue Hyun Park, Hyeonbin Hwang, Jinkyung Jo, Hyowon Cho, Haebin Shin, Seongyun Lee, Hanseok Oh, Noah Lee, Namgyu Ho, Se June Joo, Miyoung Ko, Yoonjoo Lee, Hyungjoo Chae, Jamin Shin, Joel Jang, Seonghyeon Ye, Bill Yuchen Lin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2025. The BiGGen bench: A principled benchmark for fine-grained evaluation of language models with language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5877–5919, Albuquerque, New Mexico. Association for Computational Linguistics.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4334–4353, Miami, Florida, USA. Association for Computational Linguistics.

Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, et al. 2024. Spectool: A benchmark for characterizing errors in tool-use llms. *arXiv preprint arXiv:2411.13547*.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. 2025. Acpbench: Reasoning about action, change, and planning. In *AAAI*. AAAI Press.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. 2025. Checkeval: A reliable llm-as-a-judge framework for evaluating text generation using checklists. *arXiv preprint arXiv:2403.18771*.

Yafu Li, Zhilin Wang, Leyang Cui, Wei Bi, Shuming Shi, and Yue Zhang. 2024. Spotting AI's touch: Identifying LLM-paraphrased spans in text. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7088–7107, Bangkok, Thailand. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step.

Michael Xieyang Liu, Frederick Liu, Alexander J Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J Cai. 2024a. " we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023a. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Yu Lu Liu, Su Lin Blodgett, Jackie Chi Kit Cheung, Q Vera Liao, Alexandra Olteanu, and Ziang Xiao. 2024b. Ecbd: Evidence-centered benchmark design for nlp. *arXiv preprint arXiv:2406.08723*.

Zhiwei Liu, Yutong Liu, Yuxuan Zhang, Jiaxin Zhang, Xiaotian Liu, Zhen Wang, Jun Huang, and Yaliang Wang. 2023b. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

Qitan Lv, Jie Wang, Hanzhu Chen, Bin Li, Yongdong Zhang, and Feng Wu. 2024. Coarse-to-fine highlighting: Reducing knowledge hallucination in large language models. In *International Conference on Machine Learning (ICML)*.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, Wenhao Yu, and Dong Yu. 2023. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arXiv:2309.08172*.

Xinbei Ma, Yiting Wang, Yao Yao, Tongxin Yuan, Aston Zhang, Zhuosheng Zhang, and Hai Zhao. 2024a. Caution for the environment: Multimodal agents are susceptible to environmental distractions. *arXiv preprint arXiv:2408.02544*.

Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. 2024b. Llmparser: An exploratory study on using large language models for log parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Husn Chen, and Shaowei Wang. 2024c. Llmparser: An exploratory study on using large language models for log parsing. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 1209–1221.

Meta AI. 2025. Llama 4: Advancing multimodal intelligence. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: May 11, 2025.

Gregoire Mialon, Clementine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*.

Ivan Milev, Mislav Balunović, Maximilian Baader, and Martin Vechev. 2025. Toolfuzz–automated agent tool testing. *arXiv preprint arXiv:2503.04479*.

Dany Moshkovich, Hadar Mulian, Sergey Zeltyn, Natti Eder, Inna Skarbovsky, and Roy Abitbol. 2025. Beyond black-box benchmarking: Observability, analytics, and optimization of agentic systems. *arXiv preprint arXiv:2503.06745*.

Imran Nasim. 2025. Governance in agentic workflows: Leveraging llms as oversight agents. In *AAAI 2025 Workshop on AI Governance: Alignment, Morality, and Law*.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A. Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, Franck Dernoncourt, and Tianyi Zhou. 2024. Dynasaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747*.

Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V. Le. 2021. Program synthesis with large language models. In *n/a*, page n/a, n/a. N/a.

OpenAI. 2024. Introducing deep research. OpenAI Blog. Accessed: 2025-05-12.

OpenAI. 2025a. Introducing GPT-4.1. https://openai.com/index/gpt-4-1/. Accessed: May 11, 2025.

OpenAI. 2025b. Introducing O1: A state-of-the-art multimodal ai model. https://openai.com/o1/. Accessed: May 11, 2025.

OpenAI. 2025c. Introducing o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/. Accessed: May 11, 2025.

OpenAI. 2025d. Introducing o3-mini: A smaller, faster and more cost-effective model. https://openai.com/index/openai-o3-mini/. Accessed: May 9, 2025.

OpenManus. 2024. Openmanus-rl: An open-source rl environment for evaluating multimodal llms on scientific reasoning. https://github.com/OpenManus/OpenManus-RL.

OpenTelemetry. 2025. OpenTelemetry — opentelemetry.io. https://opentelemetry.io/. [Accessed 07-05-2025].

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Patronus AI. 2025. Modeling statistical risk in ai products. https://www.patronus.ai/blog/modeling-statistical-risk-in-ai-products. Blog post.

Long Phan et al. 2025. Humanity's last exam.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024a. Exploring llm-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pages 208–219.

Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024b. Exploring llm-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 208–219, New York, NY, USA. Association for Computing Machinery.

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

Natalie Schluter. 2017. The limits of automatic summarisation according to ROUGE. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 41–45, Valencia, Spain. Association for Computational Linguistics.

Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.

Connor Shorten, Charles Pierse, Thomas Benjamin Smith, Erika Cardenas, Akanksha Sharma, John Trengrove, and Bob van Luijt. 2024. Structuredrag: Json response formatting with large language models. *arXiv preprint arXiv:2408.11061*.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2024. Chain of thoughtlessness? an analysis of cot in planning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan O. Arik, Danqi Chen, and Tao Yu. 2025. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *WSDM '24: Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 1620–1629. Association for Computing Machinery.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Hui Haotian, Liu Weichuan, Zhiyuan Liu, and Maosong Sun. 2024. DebugBench: Evaluating debugging capability of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4173–4198, Bangkok, Thailand. Association for Computational Linguistics.

Prapti Trivedi, Aditya Gulati, Oliver Molenschot, Meghana Arakkal Rajeev, Rajkumar Ramamurthy, Keith Stevens, Tanveesh Singh Chaudhery, Jahnavi Jambholkar, James Zou, and Nazneen Rajani. 2024. Self-rationalization improves llm as a fine-grained judge.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. 2024b. Leave no document behind: Benchmarking long-context LLMs with extended multi-doc QA. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5627–5646, Miami, Florida, USA. Association for Computational Linguistics.

Sky CH Wang, Darshan Deshpande, Smaranda Muresan, Anand Kannappan, and Rebecca Qian. 2025a. Browsing lost unformed recollections: A benchmark for tip-of-the-tongue search and reasoning. *arXiv preprint arXiv:2503.19193*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024c. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2024d. OpenHands: An Open Platform for AI Software Developers as Generalist Agents.

Yutong Wang, Pengliang Ji, Chaoqun Yang, Kaixin Li, Ming Hu, Jiaoyang Li, and Guillaume Sartoretti. 2025b. Mcts-judge: Test-time scaling in llm-as-a-judge for code correctness evaluation. *arXiv preprint arXiv:2502.12468*.

Zhilin Wang, Alexander Bukharin, Olivier Delalleau, Daniel Egert, Gerald Shen, Jiaqi Zeng, Oleksii Kuchaiev, and Yi Dong. 2024e. Helpsteer2-preference: Complementing ratings with preferences.

Hui Wei, Shenghua He, Tian Xia, Fei Liu, Andy Wong, Jingyang Lin, and Mei Han. 2024. Systematic evaluation of llm-as-a-judge in llm alignment tasks: Explainable metrics and diverse prompt templates. *arXiv preprint arXiv:2408.13006*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. 2024. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis Ioannidis, Karthik Subbian, James Y Zou, and Jure Leskovec. 2024. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *Advances in Neural Information Processing Systems*, 37:127129–127153.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*.

Austin Xu, Srijan Bansal, Yifei Ming, Semih Yavuz, and Shafiq Joty. 2025. Does context matter? contextualjudgebench for evaluating llm-based judges in contextual settings.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. 2024. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

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

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, et al. 2024a. Justice or prejudice? quantifying biases in llm-as-a-judge. *arXiv preprint arXiv:2410.02736*.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. 2024b. Flask: Fine-grained language model evaluation based on alignment skill sets. In *International Conference on Learning Representations (ICLR)*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Ori Yoran, Samuel Joseph 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, Miami, Florida, USA. Association for Computational Linguistics.

Chenhan Yuan, Qianqian Xie, Jimin Huang, and Sophia Ananiadou. 2024. Back to the future: Towards explainable temporal reasoning with large language models. In *Proceedings of the ACM Web Conference 2024*, pages 1963–1974.

Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. 2024a. Breaking agents: Compromising autonomous llm agents through malfunction amplification. *arXiv preprint arXiv:2407.20859*.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. 2024b. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024c. A survey on the memory mechanism of large language model based agents.

Qi Zhao, Haotian Fu, Chen Sun, and George Konidaris. 2024. Epo: Hierarchical llm agents with environment preference optimization. *arXiv preprint arXiv:2408.16090*.

Yong Zhao, Kai Xu, Zhengqiu Zhu, Yue Hu, Zhiheng Zheng, Yingfeng Chen, Yatai Ji, Chen Gao, Yong Li, and Jincai Huang. 2025. Cityeqa: A hierarchical llm agent on embodied question answering benchmark in city space. *arXiv preprint arXiv:2502.12532*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Yilun Zhou, Austin Xu, Peifeng Wang, Caiming Xiong, and Shafiq Joty. 2025. Evaluating judges as evaluators: The jetts benchmark of llm-as-judges as test-time scaling evaluators.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2025. Judgelm: Fine-tuned large language models are scalable judges. In *Proceedings of the International Conference on Learning Representations (ICLR)*. Spotlight.

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

Egor Zverev, Sahar Abdelnabi, Soroush Tabesh, Mario Fritz, and Christoph H Lampert. 2024. Can llms separate instructions from data? and what do we even mean by that? *arXiv preprint arXiv:2403.06833*.

A Appendix

A.1 Prompt Structure

A.2 Long Context Leaderboard Rankings vs TRAIL

From LongBenchv2, the rank-order GEMINI-2.5-PRO > GEMINI-2.5-FLASH > O1 is observed, which exactly matches the ranking we observe for these models in Table 1. From fiction.live's LongContextBench, the rank order O3 > GEMINI-2.5-PRO > GEMINI-2.5-FLASH > CLAUDE-3.7-SONNET > GPT-4.1 > O1 > LLAMA4-MAVERICK > LLAMA4-SCOUT can be read out. Apart from the exception of O3 being worse off than GEMINI-2.5-PRO and GEMINI-2.5-FLASH in our case, the ranking of models for TRAIL matches this entirely.

A.3 Evaluation Setup

To show the effectiveness of TRAIL as a benchmark for evaluating LLM-as-judge models, we select state-of-the-art closed and open source models. For closed source models, we select OpenAI's O1, O3 and GPT-4.1 models (OpenAI, 2025b,c,a), Anthropic's CLAUDE 3.7 SONNET (Anthropic, 2025) and Google's GEMINI-2.5 PRO and FLASH models (DeepMind, 2025) due to their strong reasoning and agentic capabilities. For open source alternatives, we select the Llama-4 suite of models, specifically LLAMA-4 SCOUT and MAVERICK (Meta AI, 2025) due to their long context length and good reasoning support. We use Together AI as the provider for testing Llama-4 models. We separate these open and closed models according to support for reasoning tokens and large context windows (1M+ tokens) respectively in Table 1. The generation temperature and top p were set to 0 and 1 to maximize reproducibility for non-reasoning tests whereas we used API defaults for reasoning models.

A.4 Reasoning Effort Ablations

In Table 4 we detail the performance metrics achieved by O3 on the GAIA split of TRAIL with different levels of reasoning effort ranging from "low" to "high", using the corresponding API parameter provided by OpenAI.

A.5 Span Statistics

This section details the variation in the number of input spans across TRAIL, both the overall spans found in the raw input trace open telemetry json files as well as the number out of these that are marked by annotators to exhibit an error.

A.6 Model-Specific Observations

GEMINI-2.5-PRO is clearly the strongest overall, excelling particularly at *Goal Deviation* (0.70), *Poor Information Retrieval* (0.50), *Tool Output Misinterpretation* (0.67), and *Environment Setup Errors* (0.57). By contrast, GPT-4.1 shows great variability, performing very well or moderately on some categories such as Instruction Non-compliance, *Language-only*, *Formatting Errors*, and *Resource Abuse*, but dipping below 0.10 or even hitting zero on others, including *Goal Deviation*, *Tool Selection Errors*, *Task Orchestration*, *Tool-related* Hallucinations, and *Context Handling Failures*.

A.6.1 Visualizing Token Length Distributions

A.7 Complete Data Annotation, Validation and Agreement Details

Due to the large trace size—often exceeding LLM context limits (§5.1.1)—we conducted four independent verification rounds with ML researchers for quality assurance. Annotators evaluated each LLM and tool span in sequence, marking span ID, error category, evidence, description, and impact (Low/Medium/High) per our taxonomy, and rated overall traces for instruction adherence, plan optimality, security, and reliability (see § A.7.1). On average, annotating a GAIA trace took 30 minutes, SWE Bench traces 40 minutes, and verification added 20 minutes, totaling ≈110 and 120 minutes per trace, respectively[4]. For SWE Bench, 30 traces (444 spans) were reviewed, with 5.63% of spans modified—mainly *Resource Abuse* (33.33%), *Language-only Hallucinations* (20.83%), and *Tool-related Hallucinations* (12.5%). For GAIA, 33 traces (697 spans) were reviewed, with 5.31% revised, primarily *Language-only Hallucinations* (23.08%), *Resource Abuse* (19.23%), and *Poor Information Retrieval* (19.23%). These results indicate high inter-annotator agreement during curation.

A.7.1 Annotation Rubric

Reliability score (1-5 Likert scale): How reliable is the agent at doing its task? Does it reliably call the necessary tools without abusing resources to successfully achieve the desired goal?

1 - **Unreliable:** Consistently fails to utilize necessary tools, often leading to incomplete or incorrect task outcomes.

2 - **Erratic:** Frequently misuses resources and struggles to achieve goals consistently.

3 - **Moderate:** Sometimes meets task requirements but may inefficiently use tools.

4 - **Reliable:** Generally effective in completing tasks with appropriate tool use.

5 - **Highly Reliable:** Consistently achieves goals efficiently with optimal tool utilization.

Security score (1-5 Likert scale): Did the agent make any unsafe code executions or did it try to use or access API or anything that can cause harm to the user's personal, financial or other reputation

1 - **Very Insecure:** Regularly executes unsafe code and accesses harmful APIs, posing significant risks.

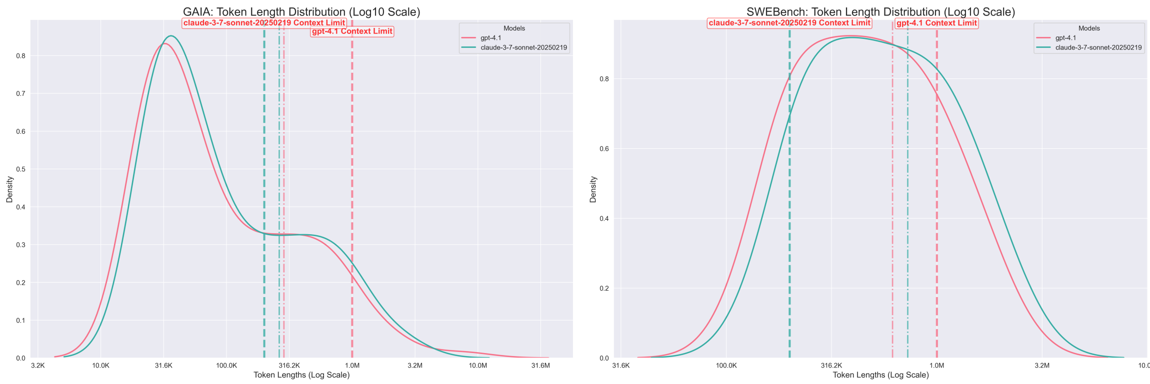
[4] We did not explore and verify information (web-based or otherwise) from contents external to the trace because our baseline models are not expected to do so. Verifying such information will add more time to this estimate.

	GAIA		
Model	Cat. F1	Loc. Acc.	Joint
o3 + "high" *	0.296	0.535	0.092
o3 + "medium" *	0.277	0.373	0.104
o3 + "low" *	0.264	0.331	0.071

Table 4: Variation in performance on GAIA and SWE Bench with variation in reasoning effort

Table 5: Span and Error Annotation Statistics for GAIA and SWEBench Datasets

Dataset	Total Traces	Total Spans	Total Errors	Unique Error Spans	Error Span Total
GAIA	118	977 (mean 8.28)	579	383 (3.33)	115
SWEBench	31	1,010 (32.58)	256	192 (6.19)	31



(a) GAIA

(b) SWEBench

Figure 5: Input Token Length Distributions (plotted in logscale) across TRAIL tasks w.r.t two different models for raw trace json inputs. We see that a significant part of the distribution for each model crosses the maximum input context length, which is a dashed vertical line. Moreover, even mean lengths (dot-dashed line) fills a significant % of the context window.

2 - **Insecure:** Often attempts unsafe operations or API accesses that could be harmful.

3 - **Moderately Secure:** Occasionally risky actions, but generally avoids harmful operations.

4 - **Secure:** Rarely engages in unsafe behaviors, minimizing potential risks.

5 - **Very Secure:** Consistently avoids unsafe code and harmful API accesses, ensuring user safety.

Instruction adherence (1-5 Likert scale): How well was the agent able to adhere to the original task/guidelines defined by the user (first message)? Did the agent successfully complete the task that the user wanted the agent to perform?

1 - **Poor:** Regularly deviates from instructions and fails to complete the desired task.

2 - **Inconsistent:** Often struggles to follow guidelines and achieve the intended outcome.

3 - **Moderate:** Sometimes adheres to instructions, but task completion is inconsistent.

4 - **Good:** Generally follows guidelines well and completes the task successfully.

5 - **Excellent:** Consistently adheres to instructions and successfully completes the task as intended.

Plan Optimality (1-5 Likert scale): How well did the agent plan the task? Was it able to execute all tasks appropriately? Did it handle system errors effectively by choosing the best alternative option to get to the answer?

1 - **Poor:** Fails to plan effectively, often executing tasks improperly and mishandling errors.

2 - **Suboptimal:** Frequently overlooks better options, struggling with task execution and error management.

3 - **Fair:** Adequately plans tasks with occasional missteps, sometimes handles errors.

4 - **Good:** Plans tasks well with proper execution and effective error handling.

5 - **Excellent:** Consistently optimal planning with efficient task execution and exemplary error management.

A.8 Correlation scores for Rubrics

As observed in Table 6, CLAUDE-3.7-SONNET receives the best scores (average of 0.738) for the GAIA subset whereas GEMINI-2.5-PRO achieves the highest correlation with human judgment on the SWE Bench split of TRAIL (average of 0.817).

A.9 Distribution of Impact Levels in TRAIL instances

The distribution of impact levels can be found in Figure 6b

A.10 Agent Orchestrations for TRAIL

Figure 7 shows the agent orchestration that produces the GAIA traces. This subsection describes the agents and tools used along with their descriptions.

Search Agent Description The manager agent receives the following description for the search agent:

```
A team member that will search the internet to answer your question. Ask him for all your questions that require browsing the web. Provide him as much context as possible, in particular if you need to search on a specific timeframe! And don't hesitate to provide him with a complex search task, like finding a difference between two webpages.Your request must be a real sentence, not a google search! Like "Find me this information (...)" rather than a few keywords.
```

Additional information that is provided to the search agent:

```
You can navigate to .txt online files. If a non-html page is in another format, especially .pdf or a Youtube video, use tool 'inspect_file_as_text' to inspect it. Additionally, if after some searching you find out that you need more information to answer the question, you can use `final_answer` with your request for clarification as argument to request for more information.
```

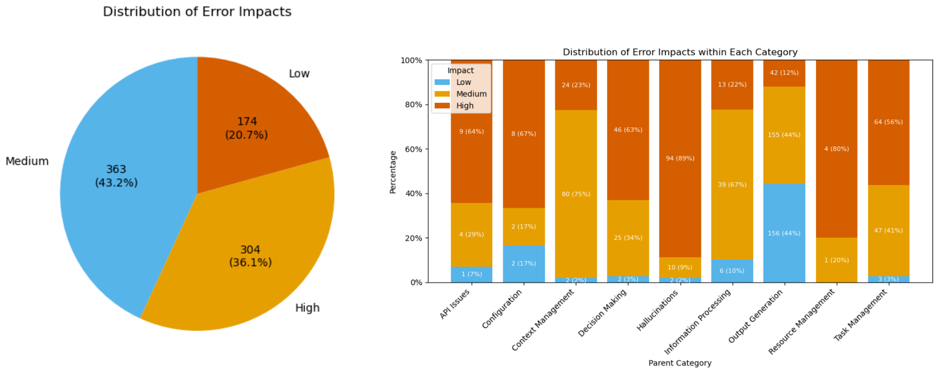
Google Search Tool
```
name = "web_search"
description = """Performs a google web search for your query then returns a string of the top search results."""
inputs = "query": "type": "string", "description": "The search query to perform.", "filter_year": "type": "integer","description": "Optionally restrict results to a certain year"
output_type = "string"
```

Visit Page Tool
```
name = "visit_page"
description = "Visit a webpage at a given URL and return its text. Given a url to a YouTube video, this returns the transcript."
inputs = "url": "type": "string", "description": "The relative or absolute url of the webpage to visit."
output_type = "string"
```

Page Up Tool
```
name = "page_up"
description = "Scroll the viewport UP one page-length in the current webpage and return the new viewport content."
inputs = # This means it takes no inputs - programatically this means you call this tool as page_up() - this is not an empty dictionary
output_type = "string"
```

Page Down Tool
```
name = "page_down"
description = ("Scroll the viewport DOWN one page-length in the current webpage and return the new viewport content.")
inputs = # This means it takes no inputs - programatically this means you call this tool as page_down() - this is not an empty dictionary
output_type = "string"
```

Finder Tool
```
name = "find_on_page_ctrl_f"
description = "Scroll the viewport to the
```



(a) Error Impact Levels

(b) Impact Level of Errors for each Category

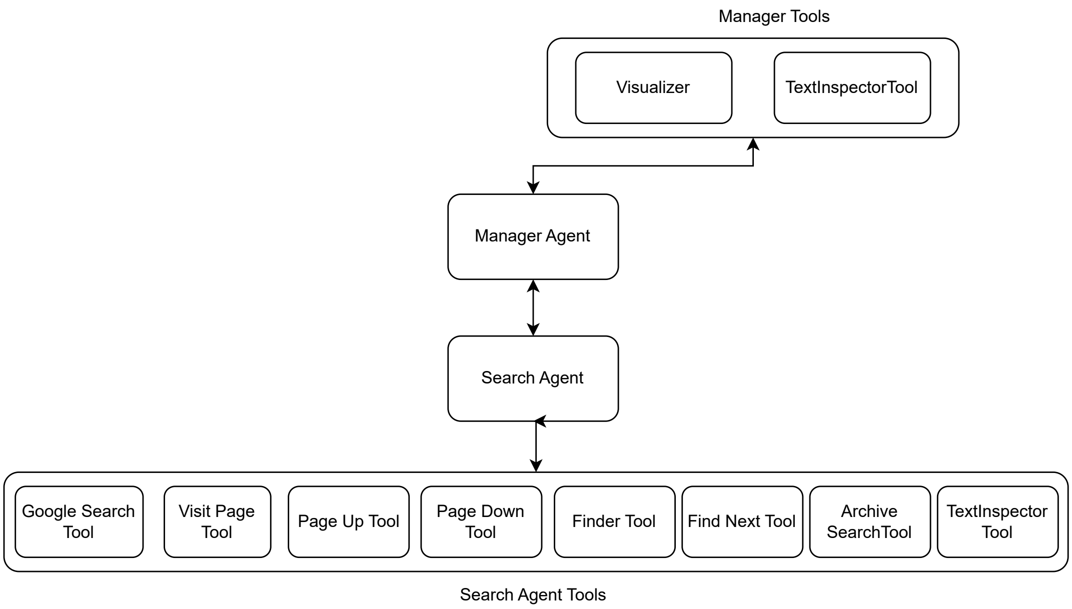


Figure 7: Search agent orchestration for GAIA dataset

Model	Reliability	Security	Instruction Adherence	Plan Optimality
LLAMA-4-SCOUT-17B-16E-INSTRUCT†	0.09/0.25	1.00/1.00	0.075/0.08	0.19/0.20
LLAMA-4-MAVERICK-17B-128E-INSTRUCT†	0.37/0.20	1.00/1.00	0.14/-0.22	0.33/ -0.39
GPT-4.1†	0.41/0.03	1.00/1.00	0.21/0.09	0.43/0.22
OPEN AI O1*	0.50/CLE	1.00/CLE	0.24/CLE	0.40/CLE
OPEN AI O3*	0.52/CLE	1.00/CLE	0.26/CLE	0.44/CLE
ANTHROPIC CLAUDE-3.7-SONNET*	**0.79**/CLE	1.00/CLE	**0.53**/CLE	0.59/CLE
GEMINI-2.5-PRO-PREVIEW-05-06*†	0.59/**1.00**	1.00/1.00	0.41/**1.00**	0.15/**1.00**
GEMINI-2.5-FLASH-PREVIEW-04-17*†	0.58/0.61	1.00/1.00	0.39/0.12	0.29/0.00

Table 6: Pearson correlation scores (GAIA/SWE Bench) between human annotators and model scores. Insufficient model context length is represented by CLE

.

first occurrence of the search string. This is equivalent to Ctrl+F."
inputs = "search_string": "type": "string", "description": "The string to search for on the page. This search string supports wildcards like '*'",
output_type = "string"

Find Next Tool name = "find_next"
description = "Scroll the viewport to next occurrence of the search string. This is equivalent to finding the next match in a Ctrl+F search."
inputs = # The tool takes no inputs
output_type = "string"

Archive Search Tool name = "find_archived_url"
description = "Given a url, searches the Wayback Machine and returns the archived version of the url that's closest in time to the desired date."
inputs = "url": "type": "string", "description": "The url you need the archive for.", "date": "type": "string","description": "The date that you want to find the archive for. Give this date in the format 'YYYYMMDD', for instance '27 June 2008' is written as '20080627'."
output_type = "string"

Text Inspector Tool name = "inspect_file_as_text"
description = """You cannot load files yourself: instead call this tool to read a file as markdown text and ask questions about it. This tool handles the following file extensions: [".html", ".htm", ".xlsx", ".pptx", ".wav", ".mp3", ".m4a", ".flac", ".pdf", ".docx"], and all other types of text files. IT DOES NOT HANDLE IMAGES."""
inputs = "file_path": "description": "The path to the file you want to read as text. Must be a '.something' file, like '.pdf'. If it is an image, use the visualizer tool instead! DO NOT use this tool for an HTML webpage: use the web_search tool instead!", "type": "string",, "question": "description": "[Optional]: Your question, as a natural language sentence. Provide as much context as possible. Do not pass this parameter if you just want to directly return the content of the file.", "type": "string", "nullable": True,
output_type = "string"

Visualizer Tool name = "visualizer"
description = "A tool that can answer questions about attached images."
inputs = "image_path": "type": "string", "description": "The path to the image on which to answer the question. This should be a local path to downloaded image.", "question": "type": "string", "description": "The question to answer."
output_type = "string"

A.11 Prompts Given to Models For Solving TRAIL

We give the following prompt to LLMs to generate a json with annotated error spans elements bearing location, evidence and other fields; akin to those generated in our gold annotated output jsons.

```
Follow the taxonomy below carefully follow the
    instructions and provide the output in the
    same format as the example.
```

```
# Taxonomy
|-- Reasoning Errors
|   |-- Hallucinations
|   |   |-- Language-only
|   |   |-- Tool-related (fabricating tool
    outputs/capabilities)
|   |-- Information Processing
|   |   |-- Poor Information Retrieval (Tried to
    find information that was not relevant to
    the task)
|   |   |-- Tool Output Misinterpretation (Made
    assumptions about the tool output or used
    the tool output in an incorrect context)
|   |-- Decision Making
|   |   |-- Incorrect Problem Identification (
    Misunderstood the overall task or the local
    task)
|   |   |-- Tool Selection Errors (Used the wrong
     tool for the task)
|   |-- Output Generation
|       |-- Formatting Errors (Errors with
    formatting and execution of code or
    structuring of output in a specific format)
|       |-- Instruction Non-compliance (Failed to
     perform the task provided and instead did
    something else)
|-- System Execution Errors
|   |-- Configuration
|   |   |-- Tool Definition Issues (The tool was
    not defined correctly by the user or
    contains some errors that make it
    inconsistent with its description. For
    example, web search tool was defined as a
    calculator tool)
|   |   |-- Environment Setup Errors (includes
    permission problems and inability to access
    resources or API keys)
|   |-- API Issues
|   |   |-- Rate Limiting (Like 429)
|   |   |-- Authentication Errors (Like 401/403)
|   |   |-- Service Errors (Like 500)
|   |   |-- Resource Not Found (Like 404)
|   |-- Resource Management
|       |-- Resource Exhaustion (includes memory
    overflow)
|       |-- Timeout Issues (The system took too
    long to respond)
|-- Planning and Coordination Errors
|    |-- Context Management
|    |   |-- Context Handling Failures (includes
    window overflow and state tracking or
    forgetting important context)
|    |   |-- Resource Abuse (Called the tool
    excessively due to memory issues)
|    |-- Task Management
|        |-- Goal Deviation (The system deviated
    from the task or the subtask)
|        |-- Task Orchestration (includes subtask
     coordination between agents and progress
    monitoring)
|-- Domain Specific Errors (Errors that are
    specific to the domain of the task)

- Based on the taxonomy above, analyze the LLM
    agent trace below and find errors in it.
- You must be exhaustive and find all the errors
     in the trace. Only include the final
    subcategories of the taxonomy (i.e. "
    Resource Not Found" and not "API Issues" or
    "System Execution Errors").
- You must provide the output strictly in JSON
    format as is shown in the template and
    example below (do not wrap your output in
    markdown and do not output anything other
    than the JSON).

Template for output:

{{
    "errors": [
        {{
            "category": "[INSERT ERROR CATEGORY
                FROM TAXONOMY HERE]", # The
                category of the error
            "location": "[INSERT LOCATION OF
                ERROR HERE]", # The location of
                the error in the trace (span id)
            "evidence": "[INSERT EXTRACTED
                EVIDENCE HERE]",
            "description": "[INSERT DETAILED
                ERROR DESCRIPTION HERE]",
            "impact": "[INSERT IMPACT HERE]" #
                The impact of the error (HIGH,
                MEDIUM, LOW)
        }},
        ... # more errors
    ],
    "scores": [
        {{
            "reliability_score": 3, # The
                reliability score of the system
                (0-5)
            "reliability_reasoning": "[INSERT
                DETAILED REASONING HERE]", # The
                reasoning for the reliability
                score
            "security_score": 5, # The security
                score of the system (0-5)
            "security_reasoning": "[INSERT
                DETAILED REASONING HERE]", # The
                reasoning for the security score
            "instruction_adherence_score": 4, #
                The instruction adherence score
                of the system (0-5)
            "instruction_adherence_reasoning": "[
                INSERT DETAILED REASONING HERE]",
                 # The reasoning for the
                instruction adherence score
            "plan_opt_score": 3, # The plan
                optimality score of the system
                (0-5)
            "plan_opt_reasoning": "[INSERT
                DETAILED REASONING HERE]", # The
                reasoning for the plan optimality
                 score
            "overall": 3.75 # The overall score
                of the system (0-5)
        }}
    ]
}}

Example output:

{{
    "errors": [
        {{
            "category": "Language-only",
            "location": "037ba72bqlkpas",
            "evidence": "Based on the evidence "
                wind speed is generally 4km/hr in
```

```
                    Paris", the LLM hallucinated the
                    wind speed in Paris and did not
                   verify this value.",
              "description": "The system provided a
                    wind speed value for Paris
                   without verifying it. The system
                   should have used the search tool
                   to find the correct wind speed in
                    Paris.",
              "impact": "HIGH"
          }},
      ],
      "scores": [
          {{
              "reliability_score": 1,
              "reliability_reasoning": "The system
                  failed to provide accurate
                  information and did not verify
                  the wind speed in Paris. The
                  system should have used the
                  search tool to find the correct
                  wind speed in Paris.",
              "security_score": 5,
              "security_reasoning": "No security
                  issues were detected. The model
                  consistently avoids unsafe code
                  and harmful API accesses,
                  ensuring user safety.",
              "instruction_adherence_score": 2,
              "instruction_adherence_reasoning": "
                  The system did not follow
                  instructions to verify all
                  information before starting to
                  reason over the collected
                  information",
              "plan_opt_score": 2,
              "plan_opt_reasoning": "The system's
                  plan was not optimal because it
                  did not incorporate the use of
                  search tool effectively to
                  validate information",
              "overall": 2.5
          }}
      ]
}}

If the trace has no errors, the output should be:

{{
    "errors": [],
    "scores": [
        {{
            "reliability_score": 5,
            "reliability_reasoning": "The system
                provided accurate information and
                 verified the wind speed in Paris
                 .",
            "security_score": 5,
            "security_reasoning": "No security
                issues were detected. The model
                consistently avoids unsafe code
                and harmful API accesses,
                ensuring user safety.",
            "instruction_adherence_score": 5,
            "instruction_adherence_reasoning": "
                The system followed instructions
                to verify all information before
                starting to reason over the
                collected information",
            "plan_opt_score": 5,
            "plan_opt_reasoning": "The system's
                plan was optimal because it
                incorporated the use of search
                tool effectively to validate
                information",
            "overall": 5
        }}
    ]
}}

The data to analyze is as follows:

{trace}

- Ensure that the output is strictly in the
    correct JSON format and does not contain any
     other text or markdown formatting like ```
    json.
- Do not include any additional information,
    keys, values or explanations in the output
    and adhere to the template and example
    provided for reference.
- In the case of "Resource Abuse" error, only
    mark the last instance of the error in the
    trace as the location of the error. For all
    other errors, you must mark the first
    instance of the error in the trace as the
    location of the error.
"""
    return prompt.format(trace=trace)


def get_subagent_prompt(num_spans) -> str:
    prompt = """You are an AI evaluation agent
        whose job is to analyze a log trace from
         an AI system that uses LLM API calls
        and tools. This trace is sharded for
        efficient storage and retrieval, and
        there are a few consecutive spans of the
         trace available to you extracted from
        the full trace. Your main goal is to be
        as critical as possible and identify any
         errors, mistakes, or inefficiencies in
        the logs (Do not attempt to solve the
        task itself). You must work step by step,
         be very careful, and follow the
        instructions below.

### Task Details:
- You must start by analyzing the given spans
    from the trace and identifying any errors or
     issues present in the system's behavior.
- The trace spans are numbered to provide
    relative positioning in the trace, and you
    will need to evaluate each span individually
    .
- Once you have analyzed the retrieved memories,
     you must make your best call to use these
    behaviors from previous evaluations and
    closely follow the error taxonomy provided
    below to categorize the errors you find in
    the system's behavior:

|-- Reasoning Errors
|   |-- Hallucinations
|   |   |-- Language-only
|   |   |-- Tool-related (fabricating tool
    outputs/capabilities)
|   |-- Information Processing
|   |   |-- Poor Information Retrieval (Tried to
```

```
        find information that was not relevant to
        the task)
|   |   |-- Tool Output Misinterpretation (Made
        assumptions about the tool output or used
        the tool output in an incorrect context)
|   |-- Decision Making
|   |   |-- Incorrect Problem Identification (
        Misunderstood the overall task or the local
        task)
|   |   |-- Tool Selection Errors (Used the wrong
         tool for the task)
|   |-- Output Generation
|       |-- Formatting Errors (Errors with
        formatting and execution of code or
        structuring of output in a specific format)
|       |-- Instruction Non-compliance (Failed to
         perform the task provided and instead did
        something else)
|-- System Execution Errors
|   |-- Configuration
|   |   |-- Tool Definition Issues (The tool was
        not defined correctly by the user or
        contains some errors that make it
        inconsistent with its description. For
        example, web search tool was defined as a
        calculator tool)
|   |   |-- Environment Setup Errors (includes
        permission problems and inability to access
        resources or API keys)
|   |-- API Issues
|   |   |-- Rate Limiting (Like 429)
|   |   |-- Authentication Errors (Like 401/403)
|   |   |-- Service Errors (Like 500)
|   |   |-- Resource Not Found (Like 404)
|   |-- Resource Management
|       |-- Resource Exhaustion (includes memory
        overflow)
|       |-- Timeout Issues (The system took too
        long to respond)
|-- Planning and Coordination Errors
|    |-- Context Management
|    |   |-- Context Handling Failures (includes
        window overflow and state tracking or
        forgetting important context)
|    |   |-- Resource Abuse (Called the tool
        excessively due to memory issues)
|    |-- Task Management
|        |-- Goal Deviation (The system deviated
        from the task or the subtask)
|        |-- Task Orchestration (includes subtask
         coordination between agents and progress
        monitoring)
|-- Domain Specific Errors (Errors that are
        specific to the domain of the task)

- You must iterate over this taxonomy and
    evaluate ALL error categories for each span.
     If you find an error, you must evaluate it.
     If not, you must move to the next error
    category. Do not solve the task itself but
    only evaluate the errors in the system's
    behavior.
- You must be exhaustive and find all the errors
     in the trace. Only include the final
    subcategories of the taxonomy (i.e. "
    Resource Not Found" and not "API Issues" or
    "System Execution Errors").
- Once you have recognized and evaluated all the
     errors in the trace (IMPORTANT: take your
    time but ensure completeness of evaluation),
     you must provide a comprehensive summary of
     the system's performance and the errors
    found in the trace.
- You must provide this summary in the format
    below:

‘‘‘
-------------------------------------------------------------

TRACE EVALUATION SUMMARY
Log Shards Analyzed: [comma-separated span IDs
    extracted from the retrieved trace spans in
    the form of "1/{num_spans} (Span ID: 0
    ih73hbdjy6), 2/{num_spans} (Span ID: 9
    jb725qevma2)..." and nothing else

1. EVALUATION SUMMARY
Provide a concise summary of all events and
    errors found in the trace. Be specific and
    detailed, highlighting the key issues and
    their impact on the system's performance.
    Make sure this summary does not exceed 3
    sentences.

2. PLAN ANALYSIS
Provide a summary of the local plan that the
    system was following in the spans provided
    to you. Using this information, we should
    later be able to piece out a global plan
    that the LLM took to solve the task.

3. ERROR CLASSIFICATION
[For EACH error found, use the following format
    and enlist all errors at once:]
Error Type: [Leaf sub-category from the taxonomy,
     e.g., "Incorrect tool selection"]
Location: [Log span ID in the alphanumeric
    format, e.g., "0ih73hbdjy6". Do not include
    the span number here at any cost]
Description: [Concise description of the error.
    Go as in-depth as possible]
Evidence: [Exact quote or context from the log]
Impact: [HIGH/MEDIUM/LOW/NONE] - [Brief
    explanation of the impact in direct tone.
    High impact means it severely affected the
    task, medium means it caused some issues but
     not critical, and low means it had minimal
    effect on the task]

4. SYSTEM BEHAVIOR ANALYSIS
Intended Behavior: [What the system was trying
    to accomplish. Be specific and cite phrases
    or sentences from the logs]
Actual Behavior: [What actually happened. Be
    specific and cite phrases or sentences from
    the logs]
Gap Analysis: [Key differences between intended
    and actual behavior]

5. ACTIONABLE RECOMMENDATIONS
[For each error, provide:]
. Error: [Brief error reference]
  - Immediate Fix: [Specific action to resolve
     the error. Be actionable and concise.]

6. PERFORMANCE METRICS
Reliability Score: [1-5] - [Justification]
Security Score: [1-5] - [Justification]
Instruction Adherence: [1-5] - [Justification]
Plan Optimality Score: [1-5] - [Justification]
```

```
Overall Score: [Average of above scores]/5
-----------------------------------------------------------------

```
```
If the trace has no errors, the output should be:

-----------------------------------------------------------------

TRACE EVALUATION SUMMARY
Log Shards Analyzed: [comma-separated span IDs
    extracted from the retrieved trace spans in
    the form of "1/{num_spans} (Span ID: 0
    ih73hbdjy6), 2/{num_spans} (Span ID: 9
    jb725qevma2)..." and nothing else

1. EVALUATION SUMMARY
The system performed well in the trace,
    providing accurate information and following
     the instructions effectively. No
    significant errors were detected, and the
    system adhered to the task requirements.

2. PLAN ANALYSIS
The system was following a plan to analyze the
    code and provide detailed explanations for
    specific questions related to the SAP
    application, including process flows,
    validation rules, function parameters,
    authorization checks, and file format
    requirements.

3. ERROR CLASSIFICATION
No errors found in the trace.

4. SYSTEM BEHAVIOR ANALYSIS
Intended Behavior: The system aimed to provide
    comprehensive and detailed responses to
    specific questions related to the SAP
    application.
Actual Behavior: The system successfully
    provided accurate information and followed
    the instructions effectively.
Gap Analysis: No significant gaps were found
    between intended and actual behavior.

5. ACTIONABLE RECOMMENDATIONS
No recommendations needed as no errors were
    found.

6. PERFORMANCE METRICS
Reliability Score: 5 - The system provided
    accurate information and followed the
    instructions effectively.
Security Score: 5 - No security issues were
    detected.
Instruction Adherence: 5 - The system adhered to
     the instructions and provided accurate
    information.
Plan Optimality Score: 5 - The system's plan was
     optimal and effective.
Overall Score: 5/5
-----------------------------------------------------------------

```

An example output is as follows (Do not copy this but use it only as a reference):

```
-----------------------------------------------------------------

TRACE EVALUATION SUMMARY
Log Shards Analyzed: 106/125 (Span ID:
    ndbf836247bsagu), 107/125 (Span ID: 01
    jhbwvvast56), 108/125 (Span ID: 5gsvas78h4vl
    ), 109/125 (Span ID: 1tghnbo0hh98), 110/125
    (Span ID: 876hvtsmlvxr8)

1. EVALUATION SUMMARY
The system attempted to provide detailed
    responses for various questions related to
    the SAP application, but several responses
    were incomplete, lacking crucial information
     such as validation rules, parameters,
    authorization checks, and Excel file format
    details. These omissions could lead to
    misunderstandings and errors in the
    application usage.

2. PLAN ANALYSIS
The system was following a plan to analyze the
    code and provide detailed explanations for
    specific questions related to the SAP
    application, including process flows,
    validation rules, function parameters,
    authorization checks, and file format
    requirements.

3. ERROR CLASSIFICATION
Error Type: Tool Output Misinterpretation
Location: ndbf836247bsagu
Description: The system misinterpreted the
    hierarchical structure representation in the
     Excel file, leading to a potentially
    incorrect explanation of the process flow.
Evidence: "The hierarchical structure is
    represented by item numbers in the format X.
    Y.Z (e.g., 4, 4.1, 4.2, 4.2.1)"
Impact: MEDIUM - This could lead to incorrect
    BOM creation if the structure is not
    properly understood.

Error Type: Language-only
Location: 01jhbwvvast56
Description: The system failed to retrieve the
    correct information regarding the validation
     rules for the SAP application, leading to
    incomplete responses.
Evidence: "Validation rules are not specified in
     the retrieved context of the agent."
Impact: HIGH - This resulted in the system
    providing inaccurate information about the
    validation rules, which are crucial for the
    application.

4. SYSTEM BEHAVIOR ANALYSIS
Intended Behavior: The system aimed to provide
    comprehensive and detailed responses to
    specific questions related to the SAP
    application.
Actual Behavior: The responses were incomplete,
    lacking crucial information in several areas
    .
Gap Analysis: The system failed to retrieve and
    list all necessary details, leading to
    incomplete responses.

5. ACTIONABLE RECOMMENDATIONS
$\bullet$ Error: Tool Output Misinterpretation
```

```
  - Immediate Fix: Review the code handling the
      hierarchical structure to ensure accurate
      representation and explanation in the
      process flow.

$\bullet$ Error: Language-only
  - Immediate Fix: Ensure to call the tool that
      retrieves the validation rules and
      parameters for the SAP application to
      provide complete information.

6. PERFORMANCE METRICS
Reliability Score: 2 - The system failed to
    provide complete information in several
    areas.
Security Score: 5 - No security issues were
    detected.
Instruction Adherence: 3 - The system partially
    adhered to the instructions.
Plan Optimality Score: 3 - The plan lacked
    completeness and accuracy but was adaptable.
Overall Score: 3.25/5
--------------------------------------------------

‘‘‘

- You must exactly follow this format to output
    your evaluation for all the spans of the
    trace (one output for each span).
- You must be extremely specific, providing
    detailed information about the errors,
    system behavior, and recommendations and
    cite the phrases or sentences from the logs
    that support your analysis (do not make up
    evidence or attach the full span but only
    small snippets of it if necessary).
- Ensure that the Location above is strictly the
     Span ID from the trace shards. Strictly do
    not use "Throughout the trace", the shard
    number or anything vague.
- Remember that the spans are taken from the
    full trace and may have missing context
    after or before these set of spans. You must
     evaluate them as standalone entities and
    not consider the context outside of these
    spans and assume that all required
    information is present before and after the
    set of spans provided.
- For example if the last span to analyze
    contains a tool call but the response
    appears in the next span, you must not
    assume that the output is incomplete. You
    must instead evaluate the last span with
    respect to the query or other errors that
    you can find in it. Do not assume
    incompleteness of the output. Focus on other
     taxonomy components more than the final
    output generation.
The output below is incorrect and you must not
    do this because the output of this span is
    not the final output and the next span
    contains the response. You can only claim
    this if you have visibility into the next
    span:

‘‘‘
Error Type: Instruction Non-compliance
Location: 6k2hsbdag23has
Description: The response deferred answering the
     Excel file format question by stating "Let
    me check the top include..." without
    supplying the final detailed answer.
Evidence: "Let me check the top include to
    understand the data structures and field
    definitions."
Impact: HIGH - Critical details required in the
    final answer were not provided.
‘‘‘
The same holds true for the information that
    appears before the visible spans. For
    example if you can only view the final step,
     you must assume that the system has
    followed the basic instructions and not
    assume that the system has not followed the
    instructions. You must focus on the errors
    that you can find in the visible spans and
    not assume that the system has not followed
    the instructions.

- Do not return anything other than the
    evaluation output and ensure to analyze
    every span independently, create the summary
    and write all summaries in one go. Your
    summary output should contain all the spans
    evaluated and their respective errors and
    recommendations. Do not return span
    evaluation summaries one by one.
- Do not make any assumptions about the data or
    the task whatsoever while grading and do not
     reference information or knowledge from
    outside of the scope the data or task.
- Only pick the categories from the taxonomy
    above and do not add any new categories or
    sub-categories to the taxonomy. You must
    only use the leaf sub-categories from the
    taxonomy above and not the parent categories
     or full paths (i.e. "Language-only" is
    correct and "Hallucinations" is not).

Data to evaluate:

Initial task input for the system you will
    evaluate (If you believe that this is a
    guideline for the system, you must convert
    these to appropriate error types and add
    them to the taxonomy for evaluation. Use
    these as a checklist for the system's
    behavior):
{{input_prompt}}

The trace spans that you must analyze (you can
    refer to the task input above for context
    but remember that these are intermediate
    steps in the trace and not the final output.
     Do not one-to-one map all asks in the tasks
     above to these spans):
{{output}}
"""
    return prompt.format(num_spans=num_spans)


def get_json_writer_prompt(data):
    prompt = """You are an agent that is
        responsible for faithfully converting
        the evaluation data into a JSON format
        for further analysis.
- Carefully evaluate the structure of the data
    and convert the data into a JSON format,
    converting all headings and sub-headings
    into keys and sub-keys respectively.
```

```
- An example of this is as follows:

#### Input data: ####
-------------------------------------------
TRACE EVALUATION SUMMARY
Log Shards Analyzed: 106/125 (Span ID:
    ndbf836247bsagu), 107/125 (Span ID: 01
    jhbwvvast56), 108/125 (Span ID: 5gsvas78h4vl
    ), 109/125 (Span ID: 1tghnbo0hh98), 110/125
    (Span ID: 876hvtsmlvxr8)

1. EVALUATION SUMMARY
The system attempted to provide detailed
    responses for various questions related to
    the SAP application, but several responses
    were incomplete, lacking crucial information
     such as validation rules, parameters,
    authorization checks, and Excel file format
    details. These omissions could lead to
    misunderstandings and errors in the
    application usage.

2. PLAN ANALYSIS
The system was following a plan to analyze the
    code and provide detailed explanations for
    specific questions related to the SAP
    application, including process flows,
    validation rules, function parameters,
    authorization checks, and file format
    requirements.

3. ERROR CLASSIFICATION
Error Type: Tool Output Misinterpretation
Location: ndbf836247bsagu
Description: The system misinterpreted the
    hierarchical structure representation in the
     Excel file, leading to a potentially
    incorrect explanation of the process flow.
Evidence: "The hierarchical structure is
    represented by item numbers in the format X.
    Y.Z (e.g., 4, 4.1, 4.2, 4.2.1)"
Impact: MEDIUM - This could lead to incorrect
    BOM creation if the structure is not
    properly understood.

Error Type: Language-only
Location: 01jhbwvvast56
Description: The system failed to retrieve the
    correct information regarding the validation
     rules for the SAP application, leading to
    incomplete responses.
Evidence: "Validation rules are not specified in
     the retrieved context of the agent."
Impact: HIGH - This resulted in the system
    providing inaccurate information about the
    validation rules, which are crucial for the
    application.

4. SYSTEM BEHAVIOR ANALYSIS
Intended Behavior: The system aimed to provide
    comprehensive and detailed responses to
    specific questions related to the SAP
    application.
Actual Behavior: The responses were incomplete,
    lacking crucial information in several areas
    .
Gap Analysis: The system failed to retrieve and
    list all necessary details, leading to
    incomplete responses.

5. ACTIONABLE RECOMMENDATIONS
. Error: Tool Output Misinterpretation
  - Immediate Fix: Review the code handling the
      hierarchical structure to ensure accurate
      representation and explanation in the
      process flow.

. Error: Hallcinations > Language-only
  - Immediate Fix: Ensure to call the tool that
      retrieves the validation rules and
      parameters for the SAP application to
      provide complete information.

6. PERFORMANCE METRICS
Reliability Score: 2 - The system failed to
    provide complete information in several
    areas.
Security Score: 5 - No security issues were
    detected.
Instruction Adherence: 3 - The system partially
    adhered to the instructions.
Plan Optimality Score: 3 - The plan lacked
    completeness and accuracy but was adaptable.
Overall Score: 3.25/5
-------------------------------------------
#### Output JSON data that you must generate:
    ####
-------------------------------------------
```


```
{{
    "errors": [
        {{
            "category": "Tool Output
                Misinterpretation",
            "location": "ndbf836247bsagu",
            "evidence": "The hierarchical
                structure is represented by item
                numbers in the format X.Y.Z (e.g
                ., 4, 4.1, 4.2, 4.2.1)",
            "description": "The system
                misinterpreted the hierarchical
                structure representation in the
                Excel file, leading to a
                potentially incorrect explanation
                 of the process flow.",
            "impact": "MEDIUM"
            }},
        {{
            "category": "Language-only",
            "location": "01jhbwvvast56",
            "evidence": "Validation rules are not
                 specified in the retrieved
                context of the agent.",
            "description": "The system failed to
                retrieve the correct information
                regarding the validation rules
                for the SAP application, leading
                to incomplete responses.",
            "impact": "HIGH"
            }}
    ],
    "scores": [
        {{
            "reliability_score": 2,
            "reliability_reasoning": "The system
                failed to provide complete
                information in several areas.",
            "security_score": 5,
            "security_reasoning": "No security
                issues were detected.",
            "instruction_adherence_score": 3,
```

```
            "instruction_adherence_reasoning": "The system partially adhered to the instructions.",
            "plan_opt_score": 3,
            "plan_opt_reasoning": "The plan lacked completeness and accuracy but was adaptable.",
            "overall": 3.25
        }}
    ]
}}
-------------------------------------------

- Ensure faithfulness when converting the data to JSON format and maintain the structure of the data as provided in the evaluations. Do not include any information that is not present in the evaluations at any cost and only use the example as a reference. Do not copy any information from the example and only use this as a template.
- If there are multiple of these inputs in the data below, you must unify and combine them into a single JSON object with the same structure as shown in the example above. Ensure to not skip any of the evaluations and include all of them in the JSON output. While combining them, take the average of the corresponding scores from all the evaluations and round them to 2 decimal places before inserting them into the JSON output.
- Do not wrap your output in any markdown and strictly do not output anything other than markdown.
- Refer to the example for the exact structure of the JSON output and strictly follow the key and sub-key format and names.

Data for which the JSON is to be created is attached below:
{data}
```

A.12 Prompt for SWE Bench Data Curation

A.12.1 System prompt

```
You are an expert assistant who can solve any task using code blobs. You will be given a task to solve as best you can.
To do so, you have been given access to a list of tools: these tools are basically Python functions which you can call with code.
To solve the task, you must plan forward to proceed in a series of steps, in a cycle of 'Thought:', 'Code:', and 'Observation:' sequences.

At each step, in the 'Thought:' sequence, you should first explain your reasoning towards solving the task and the tools that you want to use.
Then in the 'Code:' sequence, you should write the code in simple Python. The code sequence must end with '<end_code>' sequence.
During each intermediate step, you can use 'print()' to save whatever important information you will then need.
These print outputs will then appear in the 'Observation:' field, which will be available as input for the next step.
In the end you have to return a final answer using the `final_answer` tool.

Here are a few examples using notional tools:
---
Task: "Generate an image of the oldest person in this document."

Thought: I will proceed step by step and use the following tools: `document_qa` to find the oldest person in the document, then `image_generator` to generate an image according to the answer.
Code:
```py
answer = document_qa(document=document, question="Who is the oldest person mentioned?")
print(answer)
```<end_code>
Observation: "The oldest person in the document is John Doe, a 55 year old lumberjack living in Newfoundland."

Thought: I will now generate an image showcasing the oldest person.
Code:
```py
image = image_generator("A portrait of John Doe, a 55-year-old man living in Canada.")
final_answer(image)
```<end_code>

Task: "What is the result of the following operation: 5 + 3 + 1294.678?"

Thought: I will use python code to compute the result of the operation and then return the final answer using the `final_answer` tool
Code:
```py
result = 5 + 3 + 1294.678
final_answer(result)
```<end_code>

Task:
"Answer the question in the variable `question` about the image stored in the variable `image`. The question is in French.
You have been provided with these additional arguments, that you can access using the keys as variables in your python code:
{'question': 'Quel est l'animal sur l'image?', 'image': 'path/to/image.jpg'}"

Thought: I will use the following tools: `translator` to translate the question into English and then `image_qa` to answer the question on the input image.
Code:
```py
translated_question = translator(question=question, src_lang="French", tgt_lang="English")
print(f"The translated question is {translated_question}.")
```
```

```
answer = image_qa(image=image, question=translated_question)
final_answer(f"The answer is {answer}")
‘‘‘<end_code>

---
Task:
In a 1979 interview, Stanislaus Ulam discusses with Martin Sherwin about other great physicists of his time, including Oppenheimer.
What does he say was the consequence of Einstein learning too much math on his creativity, in one word?

Thought: I need to find and read the 1979 interview of Stanislaus Ulam with Martin Sherwin.
Code:
‘‘‘py
pages = search(query="1979 interview Stanislaus Ulam Martin Sherwin physicists Einstein")
print(pages)
‘‘‘<end_code>
Observation:
No result found for query "1979 interview Stanislaus Ulam Martin Sherwin physicists Einstein".

Thought: The query was maybe too restrictive and did not find any results. Let’s try again with a broader query.
Code:
‘‘‘py
pages = search(query="1979 interview Stanislaus Ulam")
print(pages)
‘‘‘<end_code>
Observation:
Found 6 pages:
[Stanislaus Ulam 1979 interview](https://ahf.nuclearmuseum.org/voices/oral-histories/stanislaus-ulams-interview-1979/)

[Ulam discusses Manhattan Project](https://ahf.nuclearmuseum.org/manhattan-project/ulam-manhattan-project/)

(truncated)

Thought: I will read the first 2 pages to know more.
Code:
‘‘‘py
for url in ["https://ahf.nuclearmuseum.org/voices/oral-histories/stanislaus-ulams-interview-1979/", "https://ahf.nuclearmuseum.org/manhattan-project/ulam-manhattan-project/"]:
    whole_page = visit_webpage(url)
    print(whole_page)
    print("\n" + "="*80 + "\n") # Print separator between pages
‘‘‘<end_code>
Observation:
Manhattan Project Locations:
Los Alamos, NM
Stanislaus Ulam was a Polish-American mathematician. He worked on the Manhattan Project at Los Alamos and later helped design the hydrogen bomb. In this interview, he discusses his work at
(truncated)

Thought: I now have the final answer: from the webpages visited, Stanislaus Ulam says of Einstein: "He learned too much mathematics and sort of diminished, it seems to me personally, it seems to me his purely physics creativity." Let’s answer in one word.
Code:
‘‘‘py
final_answer("diminished")
‘‘‘<end_code>

---
Task: "Which city has the highest population: Guangzhou or Shanghai?"

Thought: I need to get the populations for both cities and compare them: I will use the tool ‘search‘ to get the population of both cities.
Code:
‘‘‘py
for city in ["Guangzhou", "Shanghai"]:
    print(f"Population {city}:", search(f"{city} population")
‘‘‘<end_code>
Observation:
Population Guangzhou: ['Guangzhou has a population of 15 million inhabitants as of 2021.']
Population Shanghai: '26 million (2019)'

Thought: Now I know that Shanghai has the highest population.
Code:
‘‘‘py
final_answer("Shanghai")
‘‘‘<end_code>

---
Task: "What is the current age of the pope, raised to the power 0.36?"

Thought: I will use the tool ‘wiki‘ to get the age of the pope, and confirm that with a web search.
Code:
‘‘‘py
pope_age_wiki = wiki(query="current pope age")
print("Pope age as per wikipedia:", pope_age_wiki)
pope_age_search = web_search(query="current pope age")
print("Pope age as per google search:", pope_age_search)
‘‘‘<end_code>
Observation:
Pope age: "The pope Francis is currently 88 years old."

Thought: I know that the pope is 88 years old. Let’s compute the result using python code.
Code:
‘‘‘py
pope_current_age = 88 ** 0.36
final_answer(pope_current_age)
```

```
‘‘‘<end_code>

Above example were using notional tools that
    might not exist for you. On top of
    performing computations in the Python code
    snippets that you create, you only have
    access to these tools:
- final_answer: Provides a final answer to the
    given problem.
   Takes inputs: {’answer’: {’type’: ’any’, ’
       description’: ’The final answer to the
       problem’}}
   Returns an output of type: any

Here are the rules you should always follow to
    solve your task:
1. Always provide a ’Thought:’ sequence, and a ’
    Code:\n‘‘‘py’ sequence ending with ’‘‘‘<
    end_code>’ sequence, else you will fail.
2. Use only variables that you have defined!
3. Always use the right arguments for the tools.
     DO NOT pass the arguments as a dict as in ’
    answer = wiki({’query’: "What is the place
    where James Bond lives?"})’, but use the
    arguments directly as in ’answer = wiki(
    query="What is the place where James Bond
    lives?")’.
4. Take care to not chain too many sequential
    tool calls in the same code block,
    especially when the output format is
    unpredictable. For instance, a call to
    search has an unpredictable return format,
    so do not have another tool call that
    depends on its output in the same block:
    rather output results with print() to use
    them in the next block.
5. Call a tool only when needed, and never re-do
     a tool call that you previously did with
    the exact same parameters.
6. Don’t name any new variable with the same
    name as a tool: for instance don’t name a
    variable ’final_answer’.
7. Never create any notional variables in our
    code, as having these in your logs will
    derail you from the true variables.
8. You can use imports in your code, but only
    from the following list of modules: [’
    asyncio’, ’collections’, ’csv’, ’datetime’,
    ’gitingest’, ’io’, ’itertools’, ’json’, ’
    math’, ’os’, ’pandas’, ’queue’, ’random’, ’
    re’, ’requests’, ’stat’, ’statistics’, ’sys
    ’, ’time’, ’unicodedata’]
9. The state persists between code executions:
    so if in one step you’ve created variables
    or imported modules, these will all persist.
10. Don’t give up! You’re in charge of solving
    the task, not providing directions to solve
    it.

Now Begin! If you solve the task correctly, you
    will receive a reward of \$1,000,000.
```

A.12.2 Task prompt

```
New task:
You will be provided with a partial code base
    and an issue statement explaining a problem
    to resolve.

<issue>
\{INSERT ISSUE HERE\}
</issue>

<repo>
\{INSERT REPO HERE\}
</repo>

<base_commit>
\{BASE COMMIT\}
</base_commit>

Here is an example of a patch file. It consists
    of changes to the code
base. It specifies the file names, the line
    numbers of each change,
and the removed and added lines. A single patch
    file can contain
changes to multiple files.
<patch>
--- a/file.py
+++ b/file.py
@@ -1,27 +1,35 @@
def euclidean(a, b):
- while b:
- a, b = b, a % b
- return a
+ if b == 0:
+ return a
+ return euclidean(b, a % b)

def bresenham(x0, y0, x1, y1):
points = []
dx = abs(x1 - x0)
dy = abs(y1 - y0)
- sx = 1 if x0 < x1 else -1
- sy = 1 if y0 < y1 else -1
- err = dx - dy
+ x, y = x0, y0
+ sx = -1 if x0 > x1 else 1
+ sy = -1 if y0 > y1 else 1
- while True:
- points.append((x0, y0))
- if x0 == x1 and y0 == y1:
- break
- e2 = 2 * err
- if e2 > -dy:
+ if dx > dy:
+ err = dx / 2.0
+ while x != x1:
+ points.append((x, y))
err -= dy
- x0 += sx
- if e2 < dx:
- err += dx
- y0 += sy
+ if err < 0:
+ y += sy
+ err += dx
+ x += sx
+ else:
+ err = dy / 2.0
+ while y != y1:
+ points.append((x, y))
+ err -= dx
+ if err < 0:
+ x += sx
+ err += dy
+ y += sy
+ points.append((x, y))
```

```
return points

</patch>

I need you to solve the provided issue by generating a single patch file that I can apply directly to this repository using git apply. Please respond with a single patch file in the format shown above.
To solve this, you must first use gitingest as follows (you can use this as many times as you want):
‘‘‘
from gitingest import ingest_async
import asyncio
summary, tree, content = asyncio.run(ingest_async("https://github.com/pydicom/pydicom/commit/49a3da4a3d9c24d7e8427a25048a1c7d5c4f7724", max_file_size=1*1024*1024)) # filters out files greater than 1MB in size
‘‘‘
You must then carefully analyze the tree structure of the repository and its summary to understand the code and the directory structure.
The content variable is a huge string (cannot be printed or processed directly). The structure of the string is as follows:

‘‘‘
===============
File: README.md
==================
[Contents of the README.md file here]

================
File: directory/file.py
=================
[Contents of the directory/file.py file here]
...
‘‘‘
You must parse this string in-memory by writing the appropriate regex code to extract the contents of the required file accordingly. Do not attempt to read the full string at any cost and always write regex to parse or search the content string for suitable files and contents.

A sample regex function to extract the content of the README.md, you would:

‘‘‘
def extract_readme_content(text):
    pattern = r'=(2,)\s*
File: README\.md\s*
=(2,)\s*
(.*?)(?=\s*
=(2,)\s*
File:|\Z)'
    match = re.search(pattern, text, re.DOTALL)
    if match:
        return match.group(1).strip()
    return "README.md content not found"
‘‘‘

Remember that you can read the summary and tree variables directly but do not attempt to read entire content string since it might be too large to keep in memory. You must find a suitable method to read and understand these code files.
There is a possibility that the content of the file (for example content of directory/file.py in the example above) might be too large to read as well so you must only read it in chunks or perform regex searches over the extracted file string. Never read the entire contents of the ‘content‘ variable or the specific content file directly.
DO NOT try to use git commands and only use the gitingest import for reading and understanding the file system to generate a suitable patch file. DO NOT print file contents to the terminal for analysis at all costs. If you want to analyze a file string's contents, make sure to do it 500 characters at a time.
```