# DIAGNOSING WITH INSIGHTS: STRUCTURED ANALYSIS OF AGENT FAILURES VIA BEHAVIORAL ABSTRACTIONS

**Jiayi Bi[1], Yanjie Gao[2,†], Yuanmin Xie[1], Liqun Li[3], Tianyin Xu[4], Fan Yang[2], Mao Yang[2]**

[1]Tsinghua University
[2]Microsoft Research
[3]Microsoft
[4]University of Illinois Urbana-Champaign

## ABSTRACT

With the proliferation of LLM agents, the ability to understand and diagnose failures in agents is essential to achieving superior effectiveness and trustworthiness. As agent failures often manifest via long and complex trajectories, manually finding the needles in the haystack is untenable. However, traditional diagnosis techniques for software bugs can hardly address LLM agent failures, while completely relying on LLMs as the judge yields unreliable diagnosis results. To overcome these challenges, this paper presents AGENTSCOPE, a new neuro-symbolic approach for agent failure mode diagnosis. The key principle of AGENTSCOPE is to abstract agent behavior, based on its trajectories, into structured representations. Furthermore, AGENTSCOPE introduces the concept of neural invariants to specify agent behavior properties. AGENTSCOPE leverages LLM-guided reasoning atop the structured representation against neural invariants to pinpoint both the failure step and its type in the trajectory. We show the effectiveness of AGENTSCOPE on publicly available agent failure datasets (Who&When) and a more comprehensive dataset created by us (AgentErrata), where AGENTSCOPE significantly outperforms the current art in fault localization and attribution accuracy. Our work shows that integrating structured abstractions with LLM-guided reasoning enables effective, reliable, and interpretable diagnosis for agent failures.

## 1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have been driving active development of LLM agents that autonomously interact with tools and environments using natural languages. For example, agents can issue API calls, synthesize code, query databases, and cooperate with other agents to solve complex and multi-step tasks. Modern agent frameworks such as LangChain (Chase, 2022), Auto-GPT (Significant Gravitas), and OpenManus (Liang et al., 2025) orchestrate multi-step interactions, allowing agents to operate in open and dynamic environments.

Despite their exciting capabilities, LLM agents are known to fail in subtle and sophisticated ways (Cemri et al., 2025; Zhang et al., 2025c; Skalse et al., 2022; Bryan et al., 2025; Baker et al., 2025; Fu et al., 2025). Failures can occur at any step during the agent's *reasoning* (e.g., the chain of thoughts) and *action execution* (e.g., tool calls), cascade along the agent runtime behavior, and eventually manifest as specific mistake behavior or even disrupt task execution. Thus, the ability to understand and diagnose failures in agents is essential to achieving superior effectiveness and trustworthiness of agent systems. Unfortunately, as agent failures often manifest through prolonged and complex trajectories composed of numerous steps with accumulating context, manually finding needles in the haystack is slow, costly, and obviously untenable.

This work was done during Jiayi's internship at Microsoft Research and Yuanmin's internship at Microsoft.
[†]Corresponding author.

Worse still, traditional diagnosis techniques for software bugs can hardly address LLM agent failures Zeller (2009); Attariyan & Flinn (2010); Yuan et al. (2010); Zhang et al. (2019); Ren et al. (2023). The fundamental reason is that traditional techniques are confined to symbolic and logical analysis of software codes and program executions. In contrast, agent failures are often rooted in faulty reasoning errors, invalid contexts, or instruction-unfollowing behaviors across multiple steps, which involve the entanglement of fuzzy neural paradigms and rigid symbolic paradigms.

Recent studies (Zhang et al., 2025c; Cemri et al., 2025; Zhu et al., 2025; dlshriver, 2023) have proposed several neural approaches for agent failure analysis—prompting or fine-tuning LLMs with failure trajectories and asking them to identify root causes. However, our experiments show that sole LLM-based approaches often produce unreliable and incomplete diagnostic results. Even the best-performing model, GPT-5.1 (Singh et al., 2025), achieves only 18.15% accuracy on our failure attribution datasets (see Section 5.2). The cause of this limitation is that LLMs, even when fine-tuned, struggle to systematically capture multi-step reasoning and action behaviors and maintain consistent causal invariants. Thus, they are prone to confusing correlated symptoms with true causes and sensitive to contexts and instructions.

This paper presents AGENTSCOPE, a novel neuro-symbolic framework for diagnosing agent failures. The key idea of AGENTSCOPE is to abstract the agents' behavior from their trajectories into a *structured* representation, termed the Reasoning-Action Graph (ReAG), that encapsulates both the reasoning and action execution steps of the agents, enabling rigorous correctness reasoning using formally defined invariant violation conditions on their behaviors. Different from traditional program invariants (Hoare, 1969), AGENTSCOPE introduces the concept of *neural invariants*, which can be specified with neural functions to encode correctness conditions; checking such invariants requires LLM-guided reasoning. We show that LLM-guided reasoning atop structured behavior abstractions enables AGENTSCOPE to effectively diagnose the causes of the agent failure. Compared with existing vanilla LLM-as-a-judge approaches, AGENTSCOPE demonstrates stronger diagnosis ability and more interpretable results. It can accomplish both *failure localization* and *failure attribution*: the former pinpoints the root-cause step and the latter predicts the failure category.

We evaluated AGENTSCOPE on a publicly available agent failure dataset named Who&When (Zhang et al., 2025c) and a new dataset called AgentErrata created by us through failure-taxonomy-guided fault injection for more comprehensive evaluation. Results show that AGENTSCOPE achieves significant improvements in accuracy and interpretability over state-of-the-art approaches, with localization accuracies ranging from 25.40% to 77.78% on the Who&When Algorithm-Generated dataset and from 22.41% to 34.48% on the Who&When Hand-Crafted dataset, along with enhanced attribution interpretability. Moreover, AGENTSCOPE also outperforms other methods on our new dataset, AgentErrata, achieving accuracies ranging from 28.38% to 54.13%.

In summary, we make the following contributions:

- **Principle.** We show that LLM-guided reasoning atop structured representations of behavior abstractions can significantly improve agent failure diagnosis ability.
- **Concept.** We introduce the concept of neural invariants, which use neural functions to formally define and detect properties of agent misbehaviors.
- **Tooling.** We develop AGENTSCOPE, a practical runtime framework and toolchain for agent failure detection and diagnosis that is integrated with modern agent frameworks.
- **Dataset.** We create AgentErrata, a new benchmark dataset for agent failures. Generated through a failure-taxonomy-guided fault injection process, it contains a comprehensive set of failure types covering diverse agent misbehaviors.
- **Evaluation.** We present the utility of AGENTSCOPE using systematic evaluations and showcase its improved accuracy and interpretability over the current art.

## 2 BACKGROUND

Agent failures have a variety of causes through incorrect reasoning and action execution steps, in which a single failure step can cascade and lead to observable symptoms. Table 1 summarizes our taxonomy of agent failure patterns (hereafter referred to as failure modes), organized into three dimensions by where the failure manifests in an agent trajectory: **(i) Reasoning**, **(ii) Control-flow**,

and **(iii) Action**. **Reasoning** refers to failures in internal decision-making and context utilization. **Control-flow** captures failures in execution orchestration and control, including step transitions and termination. **Action** involves failures in executing decisions through external tools or environments. Reasoning failures include *Insufficient Context*, *Wrong Context*, *Instruction Unfollowing*, and *Context Miss*. Control-flow failures capture *Termination Miss*, *Premature Termination*, and *Step Loop*. Action failures include *Action Mismatch*, *Invocation Issue*, and *Execution Failure*. This taxonomy provides a comprehensive and interpretable framework for understanding and diagnosing failures of LLM-based agents. The taxonomy is derived based on our empirical analysis of failure trajectories, along with references to existing community categorizations (Cemri et al., 2025; Zhang et al., 2025c; Baker et al., 2025).

Table 1: Taxonomy of agent failure modes.

| Dimension | Category | Description |
|---|---|---|
| Reasoning | Wrong Context (WC) | Uses context that is irrelevant, invalid, or misleading, resulting in erroneous outcomes. |
| | Instruction Unfollowing (IU) | Deviates from the provided instructions or specifications, producing outputs inconsistent with task requirements. |
| | Insufficient Context (IC) | Missing essential information in the LLM input causes reasoning and decision errors. |
| | Context Miss (CM) | Overlooks relevant information from history, causing incomplete reasoning. |
| Control-flow | Termination Miss (TM) | Continues executing steps beyond task completion, causing unnecessary or infinite actions. |
| | Premature Termination (PT) | Stops execution before the task is fully completed, resulting in incomplete solutions or missing steps. |
| | Step Loop (SL) | Repeats prior steps without justification, leading to redundant or cyclical behavior. |
| Action | Action Mismatch (AM) | Selects actions (tools, APIs, etc.) that contradict its reasoning, resulting in misalignment between decision and execution. |
| | Invocation Issue (II) | Incorrectly invokes tools or APIs, producing unintended or failed operations. |
| | Execution Failure (EF) | Encounters runtime errors while executing a tool, API call, or other action, preventing successful task completion. |

# 3 METHODOLOGY

## 3.1 BEHAVIORAL ABSTRACTION

In AGENTSCOPE, an agent system's trajectory is represented as a structured graph, termed Reasoning-Action Graph (ReAG). Formally, a ReAG is a directed acyclic graph:

$$G = \{V, E\}$$

where $V$ is the set of vertices representing individual steps, and $E$ is the set of edges encoding control or data dependencies between steps. Each vertex is represented as a quadruple:

$$v_i = \langle id_i, r_i, c_i, \mathcal{I}_i \rangle$$

where $id_i$ is the unique step identifier, $r_i$ is the role of the agent performing the step, $c_i$ is the operational content, and $\mathcal{I}_i$ is the *Intermediate Semantic Representation (ISR)* of the step.

To address redundancy and long-context challenges, we construct a quickly analyzable trajectory index and memory, which motivates the design of the ISR. The ISR for each step can be formally represented as a tuple of three sub-components:

$$\mathcal{I}_i = (C_i, R_i, S_i)$$

where $C_i$ denotes the *Intent and Context* component, $R_i$ denotes the *Reasoning and Action* component, and $S_i$ denotes the *Signal and Validation* component. Each sub-component abstracts different semantic aspects of the step and supports downstream modules in interpreting, validating, and reusing information across long agent trajectories. The *Intent and Context* component $C_i$ encodes task goals, instructions, and the purpose of the step. The *Reasoning and Action* component $R_i$ captures the agent's decision-making, reasoning summaries, performed actions, external tool invocations, and outputs. The *Signal and Validation* component $S_i$ encodes internal knowledge usage, confidence, and termination signals. All three ISR components provide structured information to facilitate step identification and characterization.

Edges $e \in E$ represent control or data dependencies between steps, capturing the logical flow of reasoning and execution dynamics. Vertices are constructed through instrumentation of API calls, tool interactions, and system logs, and are further refined with semantic parsing to ensure coherent step boundaries. Even unstructured logs can thus be converted into fully annotated, structured graphs with ISR, enabling efficient reasoning, long-term memory accumulation, and downstream analysis.

### 3.2 NEURAL INVARIANTS

AGENTSCOPE introduces the concept of *neural invariants* to specify the behavioral properties of agent failures, which can be used to verify the correctness of the ReAG of a given trajectory and to diagnose failure modes. Specifically, AGENTSCOPE conducts LLM-guided reasoning to detect violations of neural invariants. When an invariant is violated, AGENTSCOPE identifies the vertex (i.e., the step) in the ReAG where the failure starts to manifest and classifies the type of the failure. Compared with vanilla LLM-as-a-judge approaches (Zhang et al., 2025c; Cemri et al., 2025), AGENTSCOPE offers several advantages: (i) precise localization of the failure step in the ReAG, (ii) fine-grained classification of the failure type based on invariant categories rather than opaque judgment criteria, (iii) highly interpretable explanations through invariant violations that explicitly expose the root causes of the failure, and (iv) more faithful and deterministic results, since the verification relies on predefined invariants rather than stochastic model judgments solely .

To enable effective failure diagnosis, we define a comprehensive set of agent behavior neural invariants based on the taxonomy of agent failures (Table 1). Each failure mode is formalized as an invariant violation; Appendix A lists all invariant violations used in AGENTSCOPE. The framework can be easily expanded to new failure mode taxonomies by adding corresponding invariant violations. Unlike traditional program invariants (Hoare, 1969), which rely solely on symbolic conditions, AGENTSCOPE supports semantic conditions via neural functions. Each neural function is implemented as a call to a general-purpose LLM (e.g., GPT-4o) with structured, task-specific prompts that incorporate ReAG information, enabling reasoning over richer context beyond raw step content. These functions are modular and can be customized, such as by using fine-tuned models. We illustrate an invariant violation with the following example:

> **Action Mismatch.** This failure occurs when an assistant's action contradicts or fails to align with the intent of the preceding reasoning step. We use $iv_f$ to denote the invariant violation associated with failure mode $f$; $iv_f = \text{true}$ indicates that the corresponding neural invariant is violated. Let $\mathbf{N}_{action} \subseteq \mathbf{N}$ denote action nodes (tool calls or outputs), and $\mathbf{N}_{reason} \subseteq \mathbf{N}$ denote reasoning nodes. For an action node $n_t$ and its preceding reasoning node $n_{t-1}$, define $\texttt{aligned}(purpose_{t-1}, action_t, tool\_name_t, tool\_args_t)$ to check whether the action type matches the intent, the tool choice is appropriate, and the action advances the task. The invariant violation is expressed as:
>
> $$iv_{mismatch} ::= \exists\, n_t \in \mathbf{N}_{action},\ n_{t-1} \in \mathbf{N}_{reason} : \\ \neg\, \texttt{aligned}(purpose_{t-1},\ action_t,\ tool\_name_t,\ tool\_args_t)$$
>
> A violation indicates misalignment between the action and its stated purpose (e.g., tool misuse, missing or incorrect actions, or hallucinated progress). When implemented with an LLM judge, `aligned()` is treated as a binary classifier.

Figure 1 shows the prompt used by the neural function `aligned()` in this invariant violation. Based on the failure mode taxonomy and corresponding invariant violation, we synthesize diverse test cases across domains to tune and validate the neural functions. We further perform manual verification to ensure their effectiveness.

### 3.3 IMPLEMENTATION

AGENTSCOPE is implemented as a modular framework for online and offline diagnosis of single-agent and multi-agent trajectories. Diagnosing raw agent traces in one shot is inherently unreliable due to the limitations of large language models (LLMs): they struggle with long contexts, are sensitive to token positions, and can lose or misinterpret implicit dependencies in sequential data. To address these challenges, AGENTSCOPE is deliberately structured into three stages, as illustrated

**Action Mismatch Neural Function `aligned()` Prompt Template**

```
You are an impartial evaluator.

Your task is to determine whether an ACTION MISMATCH occurred.

Definition:
An action mismatch occurs when the assistant's actual action does not align
with the expected action implied by the previous step.

[Previous Step Expectation]
Step purpose: "{step_purpose}"

Implied expected action: "{expected_action_description}"

[Current Step Action]
Tool called: "{tool_name}"

Tool arguments: {tool_arguments}

Action description: "{action_description}"

Evaluation criteria:
1. Does the action type match the step intent?
2. Does the tool choice align with task semantics?
3. Does the action advance the task?

Additional information: "{additional_info}"

Respond ONLY in valid JSON:
{
  "action_mismatch": true or false,
  "reason": "...",
  "mismatch_type": "tool_misuse | missing_action | action_substitution |
hallucinated_progress | none"
}
```

Figure 1: Prompt template for the action mismatch neural function `aligned()`. Placeholders (e.g., `step_purpose`, `tool_name`) are instantiated from the agent trajectory.

in Figure 2. Each stage provides a controlled representation or analysis that reduces the cognitive burden on the model while improving reliability, interpretability, and extensibility. The three stages are as follows:

**ReAG construction** converts runtime traces into a unified, annotated representation. Instrumentation captures each step, while LLMs are used selectively to parse reasoning or implicit actions. Each step is enriched with structured semantic fields—such as role, instructions, actions, missing-information signals, and step purpose—to create a stable, aligned representation. This is essential because raw traces alone are brittle: LLMs can misalign steps, forget early context, or misinterpret implicit reasoning. ReAG ensures subsequent analyses operate on explicitly aligned, semantically meaningful fields rather than raw text.

**Failure mode detection** applies a set of neural invariant checks over the annotated graph. Each check identifies the relevant trajectory steps and context and evaluates them using a task-specific judge respectively, since different failure modes may require evidence from different context ranges and trajectory locations. Candidate failures are then retained with supporting evidence rather than immediately reduced to a single diagnosis, allowing multiple hypotheses to be maintained for broader coverage and auditability while avoiding premature selection.

**Decisive error judgment** selects a single root-cause failure from the candidate set using the full trajectory and optional task context. The decisive error is defined not as the first anomaly, but as the

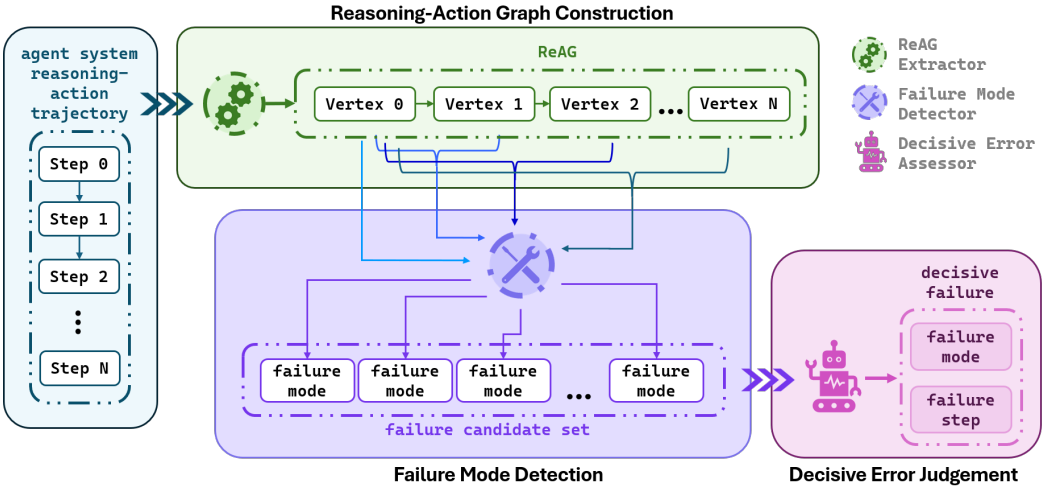


Figure 2: The workflow of AGENTSCOPE.

failure whose downstream impact most strongly explains the degraded outcome. This stage ensures that correlated or cascading violations are correctly attributed, producing a representative verdict that remains traceable to the evidence surfaced in prior stages. The strategy could be extended to identify the first failure step or the point beyond which recovery is no longer possible.

Overall, this staged design balances model-driven reasoning with structured analysis. By systematically structuring trajectories, isolating local failures, and performing controlled aggregation, the system mitigates inherent LLM limitations—context fragmentation, positional sensitivity, and retrieval/reasoning errors—while providing a faithful, inspectable, and extensible framework for trajectory diagnosis.

## 4 EXPERIMENTS

We evaluated AGENTSCOPE using both proprietary and open-source models, including GPT-4o (version: 2024-11-20) (Hurst et al., 2024), GPT-5.1 (version: 2025-11-13) (Liu et al., 2024), and DeepSeek-V3.2 (Yang et al., 2025). All models were deployed via Azure AI Foundry (Microsoft, 2025). We used a consistent hyperparameter configuration across all models, with $temperature = 0.01$ and $max\_tokens = 4096$, while keeping all other hyperparameters at their default values.

### 4.1 BENCHMARKS

We used two benchmark datasets for evaluation.

**Who&When.** The Who&When dataset (Zhang et al., 2025c) targets failure localization. It evaluates agents on the understanding towards temporal and causal relations in narrative contexts. It contains 184 human-annotated trajectories, each involving identifying who did what and when, emphasizing temporal alignment and attribution across multiple events. The Who&When dataset includes two sub-datasets that are generated from algorithm generated and hand-crafted agent systems respectively. We denote them as "Algo-Generated" and "Hand-Crafted" in §5 and evaluate the two sub-datasets separately. The former contains 126 traces and the latter contains 58 traces.

**AgentErrata.** The Who&When dataset annotates failure steps but lacks explicit failure type labels, limiting its usefulness for evaluating failure attribution. It typically marks the first step where a potential mistake arises Zhang et al. (2025c). However, as models become more fault-tolerant, early failures may be corrected by powerful agents' self-repair and therefore may not lead to final task failure (see case study in Appendix C). This can overemphasize minor mistakes while overlooking critical failures, such as repeated instructions or missing information. To address this, we

construct AgentErrata, which labels both failure steps and types and ensures that injected errors result in task failure. By using failure-taxonomy-guided fault injection, AgentErrata captures diverse failure patterns, enabling rigorous evaluation of debugging tools and highlighting weaknesses in failure detection. We select trajectories from agent frameworks and benchmark datasets represented in the current release of AgentErrata. For agent frameworks, we include OpenManus (Liang et al., 2025), OWL (Hu et al., 2025), and Mini SWE Agent (Yang et al., 2024), which span complementary paradigms of LLM-based agents, ranging from tool augmented general purpose workflows, to reasoning centric autonomous agents, and domain-specialized software engineering agents. For benchmark datasets, we draw trajectories from BrowseComp (Wei et al., 2025), SWE-Bench Lite (Jimenez et al., 2024), and WebArena (Zhou et al., 2024), which span information-seeking web browsing, real-world software engineering problem solving, and interactive web environment tasks. To create AgentErrata, we first configure the selected agents to generate execution trajectories and then filter out only those that successfully complete the tasks, thereby minimizing the impact of unlabeled and irrelevant erroneous trajectories in the dataset. We then inject failures into the original agent executions that trigger issues such as instruction unfollowing, step loops, and premature termination. These failures not only cause individual step errors but also prevent the overall task from being completed successfully, which we refer to as decisive errors. For failures that can be injected at individual steps, we employ instrumented replay to inject them at designated steps. For cases where failures involve cross-step semantic and textual dependencies, we further apply post-processing to the trajectories to identify and align such inconsistencies. Injections are applied across all categories in the current taxonomy (see Table 1) with a rough uniform distribution. To maintain contextual coherence, we additionally leverage an LLM to assess whether a given trajectory is suitable for specific categories of failure mode before the failure injection. To further ensure the quality of the dataset, all injected trajectories are manually verified, and any invalid or unreasonable trajectories are discarded. AgentErrata currently contains 303 high-quality trajectories, providing a comprehensive dataset for evaluating diagnosability of agent failures. We show the detailed distribution of AgentErrata in Appendix B.

We did not use the MAST dataset (Cemri et al., 2025), as it only provides annotations of overall failure types but lacks specific failure steps and underlying reasons. Consequently, it cannot be directly employed to reliably analyze the root causes of misattributions or to automatically evaluate the limitations of the benchmarked tools.

### 4.2 Evaluation Metrics

We use the two metrics from Zhang et al. (2025c) and introduce an additional metric to measure the effectiveness of diagnosis:

**Step-level accuracy (SLA)** quantifies the percentage of correctly identified root-cause failure steps. Formally, let $N$ be the total number of samples, $S_i$ be the true root-cause failure step for sample $i$ and $\hat{S}_i$ the predicted step, then SLA $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(\hat{S}_i = S_i\right)$, where $\mathbb{1}(\cdot)$ is the indicator function.

**Step-level accuracy with tolerance (SLAT)** considers a prediction correct if the predicted step falls within a tolerance range $\delta$ of the actual root-cause step: SLAT $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(|\hat{S}_i - S_i| \leq \delta\right)$. In other words, SLAT tolerates slight inaccuracy.

**Classification accuracy (CA)** stands for the percentage of correctly classified failure modes. Let $M_i$ be the ground truth failure mode for the $i^{th}$ trace and $\hat{\mathrm{M}}_i = \{\hat{M}_i^{(l)}\}_{l=1}^{L}$ be the predicted failure modes, where $L$ is the total number of predicted failure modes, then CA is defined as CA $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(\exists \hat{M}_i^{(l\in[1,L], l\in\mathbb{Z})} \in \hat{\mathrm{M}}_i, \hat{M}_i^{(l)} = M_i\right)$.

The Agent-level accuracy (ALA) from Zhang et al. (2025c) is not adopted because step-level metrics already reflect the responsible agent, and knowing the agent role alone does not indicate the exact failure step or cause, while agent roles are often skewed—guessing frequent agents can yield high ALA without reflecting true diagnosis ability. Thus, ALA provides little additional insight.

Table 2: SLAT (%) of the baseline methods and AGENTSCOPE across backbone models and benchmarks; higher values are better.

| Backbone | Method | AgentErrata | | | W&W Algo-Generated | | | W&W Hand-Crafted | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | T±0 | T±1 | T±3 | T±0 | T±1 | T±3 | T±0 | T±1 | T±3 |
| *Panel A: w/ Solution* | | | | | | | | | | |
| GPT-4o | W&W All-at-once | 2.14 | 6.05 | 21.00 | 14.29 | 43.65 | 69.05 | 5.26 | 8.77 | 29.82 |
| | W&W Step-by-step | 10.56 | 14.52 | 25.74 | 23.02 | **53.97** | 73.81 | 15.52 | 15.52 | 20.69 |
| | **AGENTSCOPE** | **28.38** | **34.98** | **48.51** | **31.75** | 52.38 | **77.78** | **25.86** | **29.31** | **34.48** |
| GPT-5.1 | W&W All-at-once | 2.64 | 5.61 | 18.81 | 23.81 | 48.41 | **79.37** | 3.45 | 13.79 | 25.86 |
| | W&W Step-by-step | 1.65 | 4.62 | 12.21 | **26.19** | **50.79** | 69.84 | 20.69 | **31.03** | **41.38** |
| | **AGENTSCOPE** | **29.70** | **34.65** | **50.50** | 25.40 | 44.44 | 64.29 | **22.41** | 24.14 | 29.31 |
| *Panel B: w/o Solution* | | | | | | | | | | |
| GPT-4o | W&W All-at-once | 2.09 | 4.18 | 23.69 | 16.67 | 38.89 | 69.84 | 5.17 | 10.34 | **36.20** |
| | W&W Step-by-step | 8.91 | 13.20 | 25.74 | 15.08 | 42.06 | 61.11 | 17.24 | 17.24 | 24.14 |
| | **AGENTSCOPE** | **30.03** | **35.64** | **49.83** | **28.57** | **55.56** | **77.78** | **22.41** | **24.14** | 32.76 |
| GPT-5.1 | W&W All-at-once | 3.30 | 4.62 | 18.81 | 21.43 | 46.83 | **78.57** | 3.45 | 12.07 | 29.31 |
| | W&W Step-by-step | 1.32 | 4.29 | 13.20 | **25.40** | **48.41** | 69.05 | **24.14** | **32.76** | **44.83** |
| | **AGENTSCOPE** | **31.35** | **36.63** | **54.13** | **25.40** | 45.24 | 65.87 | 22.41 | 24.14 | 29.31 |

("W&W" denotes "Who&When"; "w/ Solution" indicates that the ground-truth solution to the original task was provided during analysis, whereas "w/o Solution" indicates that it was not provided (Zhang et al., 2025c).)

# 5 RESULTS

## 5.1 HOW EFFECTIVE IS AGENTSCOPE IN FAILURE LOCALIZATION?

To evaluate the effectiveness of AGENTSCOPE in localizing the root-cause failure step, we compare it against two representative baseline approaches:

**ALL-AT-ONCE (Zhang et al., 2025c)**: This approach considers the agent's entire trajectory as a single unit and uses an LLM to directly identify failures in a single pass. While efficient, it performs holistic reasoning over the full trajectory without intermediate structured decomposition, explicit step alignment, or fine-grained attribution of responsibility across steps. As a result, failure reasoning may suffer from global entanglement of evidence, making it difficult to disentangle which specific step is causally responsible when multiple interacting errors exist.

**STEP-BY-STEP (Zhang et al., 2025c)**: This approach analyzes the trajectory in a sequential manner, where each step is evaluated together with all its preceding steps as context (i.e., prefix-conditioned reasoning). At each position, the model determines whether the current step constitutes an anomaly given the accumulated history. While this introduces local historical context, it still lacks explicit structured representation of the trajectory and does not perform graph-based aggregation or invariant-guided filtering. Consequently, it cannot maintain globally consistent dependencies across the full trajectory, and failure signals are derived from prefix-local judgments rather than global causal attribution over the entire process.

Table 2 presents the failure localization accuracy of AGENTSCOPE compared with the baseline methods across three datasets under different tolerance levels. On the AgentErrata dataset, AGENTSCOPE achieves dominant performance regardless of whether the ground-truth solution is provided. With GPT-4o (w/o Solution), AGENTSCOPE attains 30.03% at T±0, far exceeding W&W STEP-BY-STEP (8.91%) and ALL-AT-ONCE (2.09%), and the margin further increases at T±3, where AGENTSCOPE reaches 49.83% versus 25.74% and 23.69% for the baselines. With GPT-5.1, AGENTSCOPE still maintains strong performance at 31.35% (T±0, w/o Solution), compared to STEP-BY-STEP at 1.32% and ALL-AT-ONCE at 3.30%. At T±3, it reaches 54.13%, versus 13.20% and 18.81%, respectively. Interestingly, GPT-5.1 does not consistently improve failure-localization accuracy across methods. This suggests that general-purpose model capability alone does not determine diagnostic accuracy. At the same time, AGENTSCOPE's advantage on AgentErrata persists across both backbone models, consistent with the benchmark's focus on concrete failures that decisively affect task completion. Further results and analysis are provided in Section 5.3.

On the Who&When datasets, AGENTSCOPE with GPT-4o generally outperforms both baselines. For instance, on Who&When Hand-Crafted (w/ Solution), AGENTSCOPE achieves 25.86% at T$\pm$0 and 34.48% at T$\pm$3, substantially surpassing both W&W STEP-BY-STEP (15.52%, 20.69%) and W&W ALL-AT-ONCE (5.26%, 29.82%). When the ground-truth solution is not provided, AGENTSCOPE remains robust across both AgentErrata and Who&When Hand-Crafted, indicating that the structured graph representation and invariant-guided candidate preservation improve diagnostic stability under missing-context conditions. However, with GPT-5.1 on Who&When datasets, AGENTSCOPE's exact accuracy remains close to the strongest baseline. For example, Who&When STEP-BY-STEP and AGENTSCOPE both reach 25.40% on Who&When Algo-Generated (T$\pm$0, w/o Solution). This difference is therefore related to the benchmark's annotation policy rather than a uniform reduction in diagnostic effectiveness under GPT-5.1. Further inspection of the dataset reveals that some trajectories contain multiple plausible failure points. A cascading failure may include its onset, an observable manifestation, and the point at which it determines the final outcome. Who&When generally annotates the first mistake, whereas the decisive error judgment stage of AGENTSCOPE selects the candidate that most strongly explains the degraded outcome. Many reviewed GPT-5.1 predictions identify a later verification, answer-submission, or termination step on the same failure-propagation chain as the annotated onset. GPT-4o more often agrees with the earlier reference point in these comparisons, making the distinction more visible under GPT-5.1. A more detailed explanation of the case study is provided in Appendix C.

Overall, these results suggest that failure localization is not merely a step-wise anomaly detection problem, but a structured multi-stage inference task. AGENTSCOPE decomposes this task into (i) graph-based trajectory construction, (ii) invariant-guided failure-mode detection, and (iii) decisive root-cause selection. First, this design transforms open-ended diagnosis over a raw trajectory into failure-mode-specific checks over relevant steps and context. Compared with ALL-AT-ONCE and STEP-BY-STEP reasoning, it reduces the entanglement of global evidence and moves beyond prefix-local analysis. Each detected failure is also linked to localized evidence. The substantial gains on AgentErrata across both backbone models are consistent with these benefits. Second, AGENTSCOPE retains multiple detected failure modes before final selection, thereby avoiding premature commitment and separating failure detection from final attribution. The final stage could thus be extended to support different user goals, such as identifying the failure onset, a downstream manifestation, the point beyond which recovery is no longer possible, the final outcome-commitment step, or multiple steps along the failure-propagation chain. Our experiments use the current selection strategy, which prioritizes the failure with the greatest impact on the final outcome. Alternative selection strategies remain future work.

### 5.2 HOW EFFECTIVE IS AGENTSCOPE IN FAILURE ATTRIBUTION?

We further evaluate AGENTSCOPE's ability to classify failure modes beyond localizing them, a process referred to as *attribution*. Failure modes in LLM agents can manifest in diverse forms (Table 1). Existing approaches (Cemri et al., 2025; Zhang et al., 2025c) cannot simultaneously identify the failure step and classify it. We categorize failures into representative classes and assess AGENTSCOPE's ability to attribute them within these categories using the AgentErrata dataset.

Table 3 shows that AGENTSCOPE substantially improves failure-attribution accuracy on AgentErrata compared to the vanilla LLM-as-Judge baseline, which is adapted from the ALL-AT-ONCE approach (Zhang et al., 2025c) and incorporates failure mode taxonomy information in the prompt.

Table 3: Failure attribution on the AgentErrata dataset (%) across GPT-4o and GPT-5.1.

| Model | Method | SLA | CA |
|---|---|---|---|
| GPT-4o | LLM-as-Judge (w/o Solution) | 2.80 | 20.63 |
| | LLM-as-Judge (w/ Solution) | 2.14 | 19.22 |
| | **AGENTSCOPE** (w/o Solution) | 30.03 | 43.56 |
| | **AGENTSCOPE** (w/ Solution) | 28.38 | 41.58 |
| GPT-5.1 | LLM-as-Judge (w/o Solution) | 3.63 | 18.15 |
| | LLM-as-Judge (w/ Solution) | 1.98 | 19.14 |
| | **AGENTSCOPE** (w/o Solution) | 31.35 | 45.87 |
| | **AGENTSCOPE** (w/ Solution) | 29.70 | 44.88 |

("w/ Solution" indicates that the ground truth solution to the original problem that the agents were solving was given in the context during analysis, whereas "w/o Solution" denotes no ground truth solution was given (Zhang et al., 2025c).)

Beyond raw performance gains, these results suggest that failure attribution is fundamentally a *structural reasoning problem* rather than a knowledge or labeling problem. In particular, the large improvement in Step-Level Accuracy (SLA), where AGENTSCOPE achieves 28.38–31.35% compared to only 1.98–3.63% for the baseline (a 8–16$\times$ gain), indicates that monolithic LLM-as-Judge methods suffer from severe step misalignment in long-horizon trajectories. This is largely due to positional sensitivity and context compression effects, which make it difficult to consistently identify the true temporal locus of failure when reasoning is performed in a single pass over raw traces.

In contrast, AGENTSCOPE explicitly addresses this limitation through its staged design. The ReAG construction stage reduces representational ambiguity by transforming raw trajectories into semantically aligned, step-wise structured graphs, mitigating positional bias and enabling consistent cross-step reference. On top of this, the failure mode detection stage performs localized invariant checks over the structured representation, producing multiple candidate failure hypotheses instead of collapsing them into a single unstable judgment. This decomposition significantly improves recall in complex trajectories where failures may be latent, distributed, or causally entangled.

The improvement in Classification Accuracy (CA), which more than doubles from 18.15–20.63% to 41.58–45.87%, further indicates that accurate attribution requires not only correct localization but also reliable disambiguation among competing failure modes. Here, the decisive error judgment stage plays a critical role by aggregating global trajectory evidence to resolve competing hypotheses, distinguishing root causes from downstream symptoms, and preventing over-attribution to superficial anomalies. Notably, performance remains stable regardless of whether ground-truth solutions are provided, suggesting that the gains are not driven by task leakage or solution conditioning, but by structural decomposition of the attribution process itself.

### 5.3 HOW DOES AGENTSCOPE PERFORM WITH DIFFERENT BASE MODELS?

This research question investigates the influence of building AGENTSCOPE on different base LLMs. Since LLM-based agents can vary significantly in model type and size, it is important to evaluate whether AGENTSCOPE maintains high failure localization accuracy and attribution performance across these variations. We conduct experiments using representative LLMs, including GPT-4o, GPT-5.1, and DeepSeek-V3.2. For each base model, we measure AGENTSCOPE's SLA and CA in detecting failure mode, and compare the results to baseline methods. Evaluations are performed on both the Who&When and AgentErrata benchmarks.

Table 4: Failure mode attribution across different models on Who&When and AgentErrata. (%)

| Model | Method | W&W A-G | | W&W H-C | | AgentErrata | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | SLA w/o | SLA w/ | SLA w/o | SLA w/ | SLA w/o | CA w/o | SLA w/ | CA w/ |
| GPT-4o | W&W All-at-once | 16.67 | 14.29 | 5.17 | 5.26 | 2.09 | – | 2.14 | – |
| | W&W Step-by-step | 15.08 | 23.02 | 17.24 | 15.52 | 8.91 | – | 10.56 | – |
| | AGENTSCOPE | **28.57** | **31.75** | **22.41** | **25.86** | **30.03** | **43.56** | **28.38** | **41.58** |
| GPT-5.1 | W&W All-at-once | 21.43 | 23.81 | 3.45 | 3.45 | 3.30 | – | 2.64 | – |
| | W&W Step-by-step | **25.40** | **26.19** | **24.14** | 20.69 | 1.32 | – | 1.65 | – |
| | AGENTSCOPE | **25.40** | 25.40 | 22.41 | **22.41** | **31.35** | **45.87** | **29.70** | **44.88** |
| DeepSeek-V3.2 | W&W All-at-once | 26.98 | 25.40 | 3.57 | 3.51 | 0.66 | – | 0.99 | – |
| | W&W Step-by-step | 34.13 | 36.51 | 15.51 | 17.24 | 17.82 | – | 17.82 | – |
| | AGENTSCOPE | **36.51** | **38.10** | **25.86** | **24.14** | **34.98** | **45.21** | **33.33** | **44.22** |

("-" indicates that the current tool does not support this detection feature; "W&W" stands for "Who&When"; "A-G" stands for "Algo-Generated"; "H-C" stands for "Hand-Crafted"; "w/ " indicates that the ground truth solution to the original problem that the agents were solving was given in the context during analysis, whereas "w/o" denotes no ground truth solution was given (Zhang et al., 2025c).)

As shown in Table 4, AGENTSCOPE demonstrates consistently strong performance across all evaluated base models, while maintaining notably low variance compared to baseline methods.

On AgentErrata, AGENTSCOPE achieves SLA scores of 30.03% (GPT-4o), 31.35% (GPT-5.1), and 34.98% (DeepSeek-V3.2) under the w/o setting, resulting in a range of 4.95 percentage points. This relatively stable performance suggests that AGENTSCOPE effectively decouples failure localization from base model idiosyncrasies. In contrast, baseline methods exhibit strong model sensitivity: STEP-BY-STEP drops from 8.91% (GPT-4o) to 1.32% (GPT-5.1), indicating that sequential reasoning-based diagnosis is vulnerable to shifts in instruction-following behavior and internal reasoning styles across models. ALL-AT-ONCE remains consistently weak across all models, further suggesting that compressing long-horizon trajectories into a single inference step significantly limits robustness.

A key observation is that AGENTSCOPE not only improves absolute performance but also enhances cross-model invariance. Compared to the strongest baseline per model, AGENTSCOPE yields substantial gains in SLA on AgentErrata, demonstrating consistent improvements regardless of backbone architecture. This indicates that the proposed structured pipeline is largely independent of model-specific reasoning biases, instead relying on explicit trajectory analysis and invariant-based checking. A more critical insight lies in the relationship between SLA and CA. While SLA measures whether a failure is correctly localized, CA evaluates whether the root cause is correctly attributed. AGENTSCOPE exhibits a relatively stable and consistently high SLA—CA alignment across all models (e.g., 30.03% SLA vs. 43.56% CA on GPT-4o), suggesting that once a failure is detected, its causal attribution can also be reliably recovered. This implies that the intermediate representations (ReAG construction and ISR-style annotations) preserve causal signals throughout the pipeline, reducing information loss during aggregation and preventing early-stage mislocalizations from propagating into final judgments.

On the Who&When benchmark, performance exhibits a clearer structure-dependent separation. On the Hand-Crafted (H-C) subset, AGENTSCOPE achieves 22.41%–25.86% SLA (w/o), substantially outperforming ALL-AT-ONCE (3.45%–5.17%) and remaining competitive with STEP-BY-STEP (15.51%–24.14%). This suggests that trajectories in the Hand-Crafted subset exhibit deeper compositional dependencies and longer-range implicit interactions, with errors arising across multiple coupled steps rather than at isolated points. In such settings, naive aggregation or purely sequential reasoning fails to capture cross-step dependencies, while structured decomposition remains effective.

In contrast, the Algo-Generated (A-G) subset yields higher scores and smaller performance differences across methods. Trajectories in the Algo-Generated subset are generally shorter, which may make relevant failure evidence easier to isolate and partly explain the higher scores across all methods. Nevertheless, AGENTSCOPE performs well on both the Algo-Generated and Hand-Crafted subsets. An additional finding is that failure diagnosis is not monotonically correlated with model capability. Although GPT-5.1 is generally stronger in generation tasks, it does not consistently outperform GPT-4o or DeepSeek-V3.2 in diagnostic accuracy under baseline methods. This suggests

that general-purpose model capability alone does not determine diagnostic accuracy; how failure evidence is interpreted and prioritized also matters.

Overall, these results indicate that AGENTSCOPE achieves higher diagnostic accuracy than the evaluated baselines and maintains strong performance across different LLM backbones. By combining ReAG-based reconstruction with ISR-guided invariant checking, AGENTSCOPE organizes trajectory evidence into structured, localized failure modes before final attribution. This design makes the diagnostic process more transparent and auditable across heterogeneous LLM backbones.

### 5.4 WHAT IS THE RUNTIME OVERHEAD OF AGENTSCOPE?

In this section, we evaluate the runtime overhead of AGENTSCOPE. We profile AGENTSCOPE's runtime over 20 trajectories drawn from agent frameworks in AgentErrata as well as the Who&When dataset. All measurements use GPT-4o as the base model. Figure 3 shows cumulative runtime per step, with each line representing one of the 20 sampled trajectories. The per-trace runtime overhead ranges from $\sim 25\,\mathrm{s}$ (a 4-step trace with 7 calls) to $\sim 750\,\mathrm{s}$ (a 120-step trace with 242 calls). Figure 4 shows a box plot of runtime across all trajectories, grouped by the number of prompt tokens. The plot is truncated at 4k tokens to focus on the majority of cases; a small number of outliers with higher token counts and runtimes are omitted for clarity. These results indicate that AGENTSCOPE introduces a manageable runtime overhead, with latency spikes occurring only in rare cases.

Figure 5 decomposes the runtime cost by failure mode taxonomy. We observe that the *Step Loop* accounts for the largest runtime ($\sim 85\,\mathrm{s}$). This is expected, as detecting step loops failure mode requires pairwise segment comparisons, causing the number of LLM calls to scale quadratically with the number of loop-candidate windows. The next most time-consuming failure mode are *Invocation Issue* and *Execution Failure*. Each invokes $\sim 17$ calls per trace, but incurs only $\sim 2\,\mathrm{s}$ per call due to their relatively compact prompts ($\sim 375$ and $\sim 600$ tokens on average, respectively). *Wrong Context* has the highest per-call cost because its judge function processes full step context and neighboring annotations. However, it is invoked less frequently ($\sim 6$ calls per trace), keeping its total runtime moderate ($\sim 30\,\mathrm{s}$ per trace).

In general, we can conclude that the average LLM call count has a larger impact on runtime than the average prompt token count, as the time per call remains relatively stable across different failure mode checks. This implies that optimizing the number of calls—e.g., by merging multiple calls into a single call—may yield more significant runtime reductions than simply optimizing prompt length. Moreover, since the failure checks are independent of each other, they can be executed in parallel. Additionally, calls with different prompt lengths can be packed by type, further improving throughput and reducing total runtime. If cacheable prefix information is available, calls with the same prefix can be prioritized for packing, allowing subsequent requests to reuse cached key-value states and further accelerate execution. Overall, these observations suggest several potential runtime optimizations that could be applied to further improve efficiency.

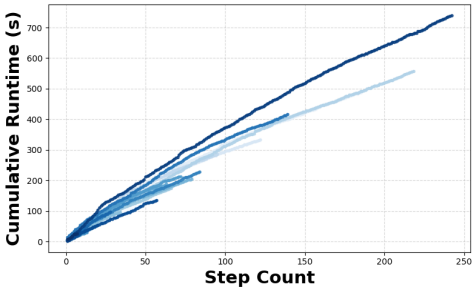


Figure 3: Cumulative runtime across steps of the 20 sampled traces.

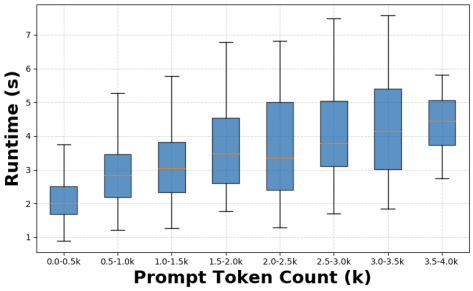


Figure 4: Time across prompt token counts of the 20 sampled traces. (x-axis truncated at 4k)

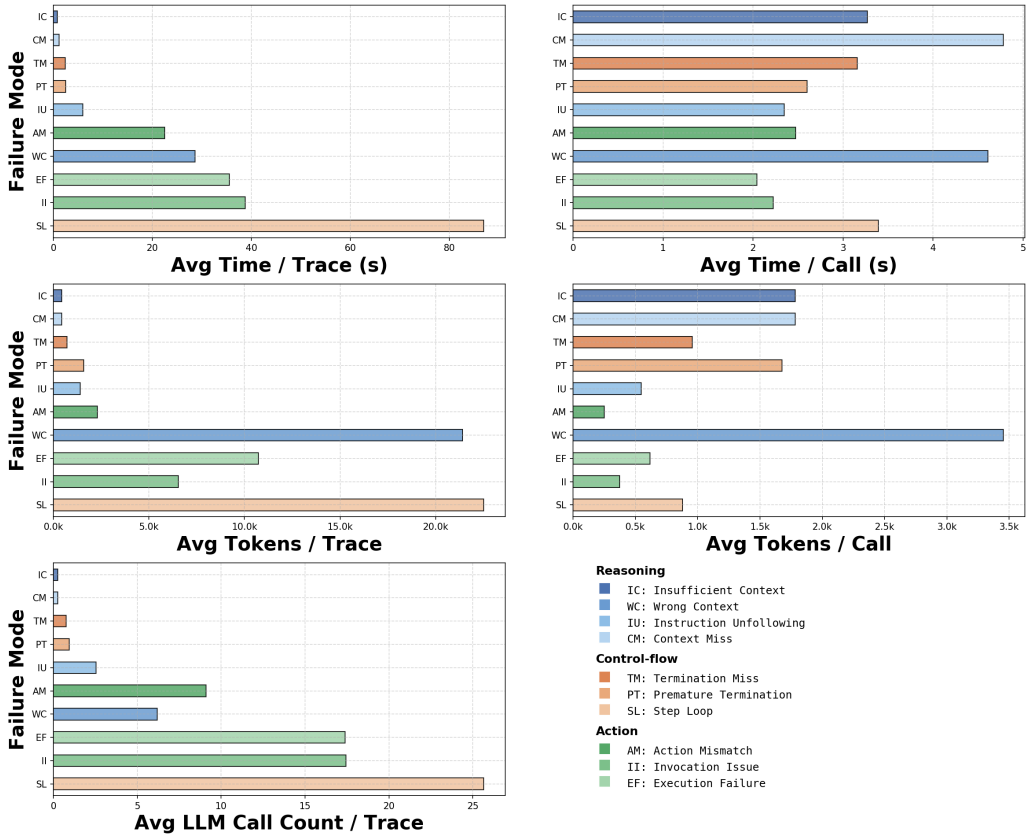


Figure 5: Time and LLM calls per invariant check across 20 sampled traces.

## 6 Related Work

**Failures of Agents.** As LLM-based agents tackle increasingly complex tasks such as planning, web navigation, and tool use, understanding their failure modes becomes critical. Prior work highlights issues like tool misuse, hallucinated goals, and premature task termination. Architectural improvements such as memory modules or hierarchical task decomposition (e.g., Voyager (Wang et al., 2023), AssistGPT (Gao et al., 2023)) primarily focus on improving task success, offering limited introspection into why failures occur.

Existing agent failure studies (Cemri et al., 2025) classify overall agent mistakes and construct failure taxonomies. In particular, Zhang et al. (2025c); Zhu et al. (2025) identify failure steps using LLMs as judges or via LLM reasoning, whereas Zhang et al. (2025a) fine-tunes models to automatically detect agent failures. Differently, AgentScope introduces a structured, interpretable, and reliable neuro-symbolic framework that encodes agent execution trajectories into *behavioral abstractions*. By representing raw trajectories in a structured form, AgentScope enables neural-invariant-guided reasoning to systematically analyze agent behavior against formally defined invariants, pinpoint the exact step at which a failure occurs, and classify its type. Since AgentScope is built on LLMs as the base models, its work is orthogonal to that of fine-tuned detection models. On the other hand, the structured analysis and design of invariants here are orthogonal to directly using LLMs as judges, and can thus benefit existing approaches.

**LLM as a Judge.** Recent works (Gu et al., 2024; Li et al., 2024) explore using LLMs as evaluators to assess output quality, correctness, or alignment. Benchmarks like Arena (Chiang et al., 2024) rank responses across tasks including QA, dialogue, and reasoning. While LLMs demonstrate meta-evaluative abilities, critiques (Chen et al., 2024; Szymanski et al., 2025) show they can be unreli-

able or inconsistent, especially for nuanced reasoning or domain-specific instructions. Compared to LLM-as-judge approaches, AGENTSCOPE leverages structured trajectory representations to guide step-wise reasoning diagnosis, offering interpretable and precise failure identification rather than holistic or subjective black-box assessments.

## 7 CONCLUSION

We presented AGENTSCOPE, a neuro-symbolic framework for diagnosing failures in LLM agents that abstracts long and heterogeneous agent trajectories into ReAGs, applies neural invariant checking to surface candidate failure modes, and selects the decisive error that best explains the final degraded outcome, enabling more localized, auditable, and interpretable diagnoses than monolithic LLM-as-a-judge approaches. Across both the public Who&When benchmark and our new benchmark AgentErrata, AGENTSCOPE achieves substantially stronger failure localization and attribution performance than prior baselines, while also supporting fine-grained classification of failure modes and suggesting that structured behavioral abstraction and invariant-guided reasoning are promising foundations for debugging increasingly complex agent systems.

## REFERENCES


Mona Attariyan and Jason Flinn. Automating Configuration Troubleshooting with Dynamic Information Flow Analysis. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, October 2010.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.

Pete Bryan, Giorgio Severi, Joris de Gruyter, Daniel Jones, Blake Bullwinkel, Amanda Minnich, Shiven Chawla, Gary Lopez, Martin Pouliot, Adam Fourney, Whitney Maxwell, Katherine Pratt, Saphir Qi, Nina Chikanov, Roman Lutz, Raja Sekhar Rao Dheekonda, Bolor-Erdene Jagdagdorj, Eugenia Kim, Justin Song, Keegan Hines, Daniel Jones, Richard Lundeen, Sam Vaughan, Victoria Westerhoff, Yonatan Zunger, Chang Kawaguchi, Mark Russinovich, and Ram Shankar Siva Kumar. Taxonomy of failure mode in agentic ai systems. White paper, Microsoft, Redmond, WA, 2025. URL `https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Taxonomy-of-Failure-Mode-in-Agentic-AI-Systems-Whitepaper.pdf`.

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

Harrison Chase. Langchain. `https://github.com/langchain-ai/langchain`, 10 2022.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. Humans or llms as the judge? a study on judgement biases. *arXiv preprint arXiv:2402.10669*, 2024.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

dlshriver. intercepts. `https://github.com/dlshriver/intercepts`, 2023. URL `https://github.com/dlshriver/intercepts`. Accessed: 2025-09-23.

Tingchen Fu, Jiawei Gu, Yafu Li, Xiaoye Qu, and Yu Cheng. Scaling reasoning, losing control: Evaluating instruction following in large reasoning models. *arXiv preprint arXiv:2505.14810*, 2025.

Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*, 2023.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12 (10):576–583, October 1969.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025. URL `https://arxiv.org/abs/2505.23885`.

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

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL `https://arxiv.org/abs/2310.06770`.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv:2411.16594*, 2024.

Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, and Xiao Tang. Openmanus: An open-source framework for building general ai agents, 2025. URL https://doi.org/10.5281/zenodo.15186407.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Microsoft. Azure ai foundry, 2025. URL https://azure.microsoft.com/en-us/products/ai-foundry. Accessed: 2025-09-24.

Xiang (Jenny) Ren, Sitao Wang, Zhuqi Jin, David Lion, Adrian Chiu, Tianyin Xu, and Ding Yuan. Relational Debugging — Pinpointing Root Causes of Performance Problems. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*, July 2023.

Significant Gravitas. AutoGPT. URL https://github.com/Significant-Gravitas/AutoGPT.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Annalisa Szymanski, Noah Ziems, Heather A Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A Metoyer. Limitations of the llm-as-a-judge approach for evaluating llm outputs in expert knowledge tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, pp. 952–966, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. SherLog: Error Diagnosis by Connecting Clues from Run-Time Logs. In *Proceedings of the 15th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS XV)*, March 2010.

Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging (2nd Edition)*. Morgan Kaufmann Publishers, June 2009.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025a.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Agents failure attribution: Who&when handcrafted case 2. https://github.com/ag2ai/Agents_Failure_Attribution/blob/main/Who%26When/Hand-Crafted/2.json, 2025b. Accessed: 2026-04-10.

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

Yongle Zhang, Kirk Rodrigues, Yu Luo, Michael Stumm, and Ding Yuan. The Inflection Point Hypothesis: A Principled Debugging Approach for Locating the Root Cause of a Failure. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*, October 2019.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Chenyang Zhu, Spencer Hong, Jingyu Wu, Kushal Chawla, Charlotte Tang, Youbing Yin, Nathan Wolfe, Erin Babinsky, and Daben Liu. Raffles: Reasoning-based attribution of faults for llm systems. *arXiv preprint arXiv:2509.06822*, 2025.

# A NEURAL INVARIANT VIOLATIONS

In this section, we present the definition of neural invariants and describe how violations of these invariants manifest in agent trajectories.

## A.1 WRONG CONTEXT

Wrong context occurs when an agent uses irrelevant, fabricated, or improperly assumed information in its reasoning—either failing to clarify genuinely ambiguous input or hallucinating prior context that does not exist.

Let $\mathbf{N}_{output} \subseteq \mathbf{N}$ be the set of output nodes and let $\mathbf{N}_{user} \subseteq \mathbf{N}$ be the set of user-input nodes. For the candidate node $n_w$ and its backward context windows, define two judging functions. $\texttt{failedToClarify}(user_t, output_w)$ evaluates whether the user input is ambiguous but the assistant proceeded without asking for clarification. $\texttt{hallucinated}(output_w, W_1, \ldots, W_m)$ evaluates whether the assistant asserts prior context or tool results that do not appear in the checked context windows $W_1, \ldots, W_m$.

The invariant violation is expressed as:

$$iv_{wrong} ::= \exists\, n_w \in \mathbf{N}_{output} : \\ \texttt{failedToClarify}(user_t,\ output_w)\ \vee\ \texttt{hallucinated}(output_w,\ W_1, \ldots, W_m).$$

Violation indicates that the agent is reasoning from an unsupported or improperly clarified contextual state, potentially leading to downstream errors. If an LLM is used as the judge, the threshold is replaced by a binary classification label.

## A.2 INSTRUCTION UNFOLLOWING

Instruction unfollowing occurs when the agent's output fails to satisfy the given instructions, including role requirements, formatting constraints, and requested behavior.

Let $\mathbf{N}_{output} \subseteq \mathbf{N}$ be the set of nodes representing final outputs, and let $\mathbf{I}$ be the set of extracted instructions from system or instruction-bearing steps. For an output node $n_f \in \mathbf{N}_{output}$ and an instruction $i \in \mathbf{I}$, define a judging function $\texttt{followed}(instruction_i, output_f)$ that evaluates whether the output satisfies the instruction's requirements.

The invariant violation is expressed as:

$$iv_{unfollowed} ::= \exists\, i \in \mathbf{I},\ n_f \in \mathbf{N}_{output} : \neg\, \texttt{followed}(instruction_i,\ output_f).$$

Violation indicates that the agent's final output does not satisfy the instructions governing that output–for example through role violation, formatting violation, missing sections, ignored instructions, or refusal without valid reason. If an LLM is used as the judge of `followed`, the threshold is replaced by a binary classification label.

## A.3 INSUFFICIENT CONTEXT

Insufficient context occurs when an agent explicitly claims that necessary information is missing, and the claimed information is indeed absent from the available context.

Let $\mathbf{N}_{claim} \subseteq \mathbf{N}$ be the set of nodes in the ReAG where the assistant claims missing or insufficient context (detected via textual hooks such as "cannot be determined", "insufficient context", or explicit `insufficient_context_flag` annotations). For a candidate node $n_c \in \mathbf{N}_{claim}$ and its backward context windows $W_1, \ldots, W_m$, define a judging function $\texttt{infoAbsent}(claim_c, W_1, \ldots, W_m)$ that evaluates whether the claimed missing information is genuinely absent from all checked context windows.

The invariant violation is expressed as:

$$iv_{insuff} ::= \exists\, n_c \in \mathbf{N}_{claim} : \texttt{infoAbsent}(claim_c,\ W_1, \ldots, W_m).$$

Violation indicates that the agent's lack-of-context claim is grounded in genuinely missing evidence—the required information truly does not exist in the provided context. If an LLM is used as the judge of `infoAbsent`, the threshold is replaced by a binary classification label.

### A.4 CONTEXT MISS

Context miss occurs when the agent claims information is missing even though it is available in the prior context, or when the agent failed to utilize available context when it was needed for reasoning or action execution.

Using the same candidate node set $\mathbf{N}_{claim}$ as the invariant of Insufficient Context, for a candidate node $n_c \in \mathbf{N}_{claim}$ and its backward context windows $W_1, \ldots, W_m$, define a judging function $\texttt{infoPresent}(claim_c, W_1, \ldots, W_m)$ that evaluates whether the claimed missing information actually exists in at least one context window, and a function $\texttt{confidentUngrounded}(output_c, W_1, \ldots, W_m)$ that evaluates whether the assistant reached a definitive conclusion without citing supporting evidence from the context.

The invariant violation is expressed as:

$$iv_{ctxmiss} ::= \exists\, n_c \in \mathbf{N}_{claim} : \\ \texttt{infoPresent}(claim_c,\ W_1, \ldots, W_m)\ \vee\ \texttt{confidentUngrounded}(output_c,\ W_1, \ldots, W_m).$$

Violation indicates that the agent mishandled available or required context rather than merely lacking it – either by ignoring present information or by asserting conclusions without grounding. If an LLM is used as the judge, the threshold is replaced by a binary classification label.

### A.5 TERMINATION MISS

Termination miss occurs when the assistant already produced the final answer but continued to act (reasoning or tool calls) instead of stopping.

Let $\mathbf{N}_{output} \subseteq \mathbf{N}$ be the set of nodes representing final outputs, and let $\mathbf{N}_{post} \subseteq \mathbf{N}$ be the set of assistant nodes occurring after the final output step. Define a judging function $\texttt{continued}(n_f, \mathbf{N}_{post})$ that evaluates whether there is any post-final-output assistant activity (tool calls, reasoning, or mixed actions).

The invariant violation is expressed as:

$$iv_{termmiss} ::= \exists\, n_f \in \mathbf{N}_{output} : |\mathbf{N}_{post}| > 0\ \wedge\ \texttt{continued}(n_f,\ \mathbf{N}_{post}).$$

Violation indicates that the agent failed to recognize task completion after producing its final answer, leading to redundant or infinite reasoning steps. If an LLM is used as the judge of `continued`, the threshold is replaced by a binary classification label.

### A.6 PREMATURE TERMINATION

Premature termination occurs when the agent stops reasoning or task execution before satisfying the required end conditions.

Let $n_{last}$ denote the last executed node in the ReAG, and $H$ denote the entire conversation history. Define $\texttt{endReq}(H)$ as the end-condition requirements extracted from the first step carrying a non-empty `end_condition_instruction` annotation, together with required format labels that the final output may need to satisfy(e.g. `Explanation`, `Exact Answer`, `Confidence`). Define a judging function $\texttt{goalMet}(output_{last}, endReq)$ that evaluates whether the final output satisfies the required end-condition content.

The invariant violation is expressed as:

$$iv_{premature} ::= \neg\, \texttt{goalMet}(output_{last},\ \texttt{endReq}(H)).$$

Violation indicates that the agent terminated before producing the required output content, resulting in incomplete solutions or missed requirements. If an LLM is used as the judge of `goalMet`, the threshold is replaced by a binary classification label.

### A.7 STEP LOOP

Step loop occurs when a consecutive group of steps repeats the same action or intent without advancing the task.

Let $\mathbf{N}$ be the set of nodes in the ReAG. For a loop window of size $k$ and a start index $t$, define two consecutive segments $A_t = (n_{t-k}, \ldots, n_{t-1})$ and $B_t = (n_t, \ldots, n_{t+k-1})$. Define a judging function $\texttt{loopRepeat}(A_t, B_t)$ that evaluates whether segment $B_t$ semantically repeats segment $A_t$ with no observable progress—by comparing roles, step purposes, actions, reasoning summaries, and tool outcomes across corresponding positions.

The invariant violation is expressed as:

$$iv_{loop} ::= \exists\, t : \texttt{loopRepeat}(A_t,\ B_t).$$

Violation indicates that the agent is trapped in repeated local behavior without moving the task forward. If an LLM is used as the judge of `loopRepeat`, the threshold of semantic similarity is replaced by a binary classification label.

### A.8 ACTION MISMATCH

Action mismatch occurs when the assistant's executed action contradicts or does not align with the intent expressed in the immediately preceding reasoning step.

Let $\mathbf{N}_{action} \subseteq \mathbf{N}$ be the set of nodes representing action steps (i.e. tool calls or output steps), and let $\mathbf{N}_{reason} \subseteq \mathbf{N}$ be the set of reasoning nodes that establish step purposes. For an action node $n_t$ and its preceding reasoning node $n_{t-1}$, define a judging function $\texttt{aligned}(purpose_{t-1}, action_t, tool\_name_t, tool\_args_t)$ that evaluates whether the action type matches the step intent, the tool choice aligns with task semantics, and the action advances the task.

The invariant violation is expressed as:

$$\begin{aligned} iv_{mismatch} ::= \exists\, n_t \in \mathbf{N}_{action},\ n_{t-1} \in \mathbf{N}_{reason} :& \\ \neg\, \texttt{aligned}(purpose_{t-1},\ action_t,\ tool\_name_t,\ tool\_args_t)& \end{aligned}$$

Violation indicates that the agent took an action that is not well aligned with its stated local purpose – for example through tool misuse, a missing action, action substitution, or hallucinated progress. If an LLM is used as the judge of `aligned`, the threshold is replaced by a binary classification label.

### A.9 INVOCATION ISSUE

Invocation issue occurs when the assistant's tool call does not comply with the tool's specification—missing required parameters, invalid types, unknown actions, or calling a tool not found in the provided tool list.

Let $\mathbf{N}_{call} \subseteq \mathbf{N}$ be the set of nodes representing tool invocations. For a node $n_t \in \mathbf{N}_{call}$, let $spec_t$ denote the matching tool schema from the tool list. Define a judging function $\texttt{schemaOK}(call_t, spec_t, response_t)$ that evaluates whether the invocation complies with the tool's parameter specification and dependencies.

The invariant violation is expressed as:

$$iv_{invocation} ::= \exists\, n_t \in \mathbf{N}_{call} : \neg\, \texttt{schemaOK}(call_t,\ spec_t,\ response_t).$$

Violation indicates that the tool invocation is not valid under the interface expected by the environment – for example through a missing required parameter, an invalid parameter type, an unknown action, incompatible parameter combinations, a tool not found in the tool list, or a tool-reported invocation error. If an LLM is used as the judge of `schemaOK`, the threshold is replaced by a binary classification label.

### A.10 Execution Failure

Execution failure occurs when the tool's response indicates that the requested operation did not complete successfully (e.g. exception, HTTP error, empty result, timeout, or permission denial).

Let $\mathbf{N}_{tool} \subseteq \mathbf{N}$ be the set of nodes associated with tool responses. For a node $n_t \in \mathbf{N}_{tool}$, define a judging function $\texttt{execFailed}(response_t)$ that evaluates whether the tool response indicates an error or non-success.

The invariant violation is expressed as:

$$iv_{exec} ::= \exists\, n_t \in \mathbf{N}_{tool} : \texttt{execFailed}(response_t).$$

Violation indicates that the requested operation did not execute as intended, potentially due to tool errors, empty results, timeouts, permission denial, rate limiting, invalid input, or network issues. If an LLM is used as the judge of `execFailed`, the threshold is replaced by a binary classification label.

## B Failure Mode Distribution of AgentErrata

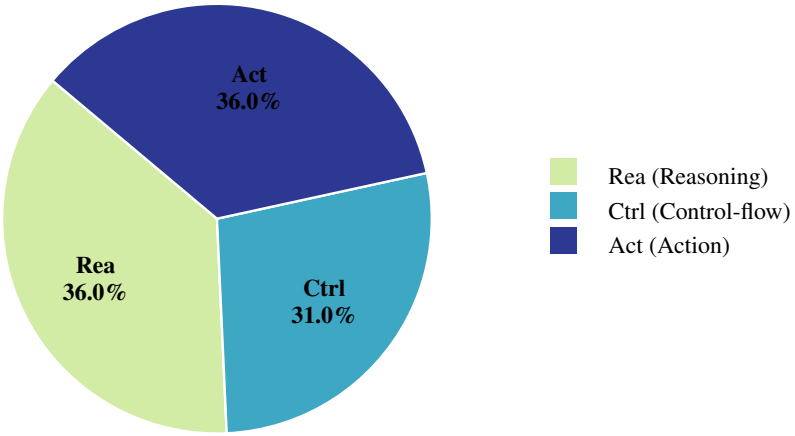


Figure 6: Failure Category Distribution of AgentErrata

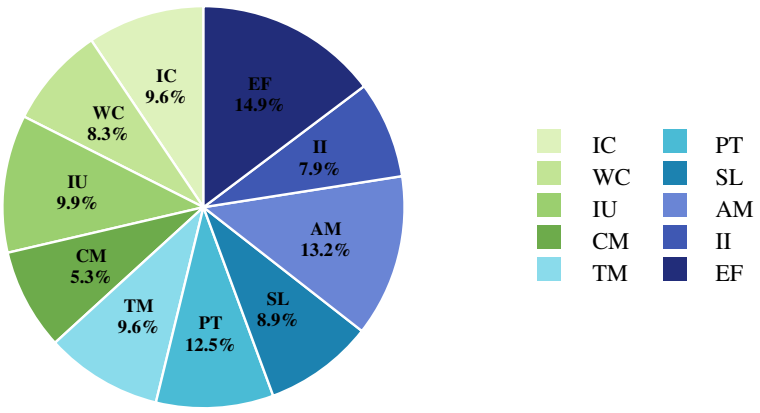


Figure 7: Failure Mode Distribution of AgentErrata

This section provides a detailed breakdown of the failure composition in AgentErrata, aiming to characterize the structural diversity of diagnostic targets. Understanding the distribution of failure

types is crucial, as it directly reflects the heterogeneity and complexity of trajectory-level errors, which in turn determines the difficulty of localization and attribution.

As shown in Figure 6, failures are broadly categorized into three high-level groups: reasoning-related (Rea), control-flow-related (Ctrl), and action-related (Act) failures. The distribution is relatively balanced across these categories, indicating that no single failure type dominates the dataset. This balance ensures that evaluation is not biased toward a specific reasoning pattern, but instead requires robust handling of diverse failure mechanisms spanning cognitive reasoning, procedural execution, and decision-level actions.

Beyond coarse-grained categorization, Figure 7 further decomposes failures into fine-grained modes, including IC, WC, IU, CM, TM, PT, SL, AM, II, and EF, with their full names listed in Table 1. It is worth noting that the slight distributional variations observed in the dataset stem from the construction process, in which we inject failures into successfully completed trajectories. During this fault injection process, the LLM is required to judge whether the injected failure mode is consistent with the context of the original trajectory, which introduces subtle biases into the resulting failure distribution. Such a distributional property is important for evaluating diagnostic systems. In particular, it prevents models from overfitting to a narrow subset of failure patterns and instead encourages more generalizable failure localization and attribution capabilities. Overall, the failure distributions in AgentErrata demonstrate both semantic diversity and structural balance, making it a suitable benchmark for evaluating robust trajectory diagnosis methods under realistic and varied failure conditions.

## C Case Study of Who&When Dataset

Traces in the Who&When dataset often contain multiple errors of varying types and severities, rather than a single decisive mistake. The annotated "mistake step" typically marks the earliest detectable issue in a trajectory, but it does not necessarily correspond to the most causally critical failure.

Consider the following example Zhang et al. (2025b). The ground-truth mistake is annotated at Step 4, where the WebSurfer retrieves only a search results page without extracting structured information (e.g., series list, number of seasons, or ratings). While this step reflects information insufficiency, it still represents a plausible intermediate exploration step and does not inherently prevent successful task completion. This interpretation is further supported by the system's internal state, which indicates continued progress at this stage.

In contrast, later steps (e.g., Steps 11 and 15) exhibit a more critical failure mode. The WebSurfer repeatedly performs the same action—clicking the same page and observing an identical viewport—without any meaningful state update or information gain. This behavior forms a stagnation loop, which is explicitly detected by the system (is_in_loop: true) and directly blocks further progress toward task completion. Notably, even after the Orchestrator issues refined instructions, the WebSurfer continues to repeat the same ineffective actions, indicating a failure to recover from the erroneous state.

Finally, the trajectory does not terminate due to successful completion or reasoning convergence. Instead, it is interrupted by a system-level failure during the orchestration phase. Specifically, a content filtering violation triggers a ResponsibleAI policy error, which results in a BadRequestError during the orchestrator's model client call for ledger updates. Consequently, execution halts prematurely.

This example illustrates that multiple failure modes can coexist within a single trajectory, spanning early-stage information insufficiency and later-stage behavioral stagnation. Crucially, the annotated mistake corresponds to the former, rather than the latter, which is more decisive for the final outcome. This limitation may reduce its effectiveness in evaluating a model's ability to localize the most impactful failure points. More broadly, effective failure localization requires not only identifying the first erroneous step, but also reasoning about its downstream impact on task completion.