FROM AGENT FAILURES TO FAULTY DIRECTIVES: ATTRIBUTION-BASED AGGREGATE ANALYSIS OF SKILL FILES

Anonymous authors
Paper under double-blind review

ABSTRACT

Current agentic systems increasingly rely on suites of *Skill* files to provide task-specific capabilities and encode proprietary knowledge that may be absent from the model's training data. Consequently, inconsistencies and errors in these files can directly affect agent behavior, leading to user dissatisfaction or task failure. However, it remains challenging for agent developers to identify which parts of a Skill file cause failures or user dissatisfaction. To the best of our knowledge, we are the first to study how agent failures can be traced to Skill directives. We introduce SKILLROOT, an execution-trace-driven framework that attributes agent actions to their supporting context and represents Skill content as a knowledge graph of directives, entities, and relationships. SKILLROOT aggregates evidence from successful and failed executions to compute a failure-association prior over directives, which an LLM-based judge combines with a new failure trace to localize responsible Skill chunks. Across our evaluation, SKILLROOT improves faulty-directive identification by up to 6% over standard baselines.

1 INTRODUCTION

Large language models (LLMs) are reshaping how software systems are designed and operated. Beyond text generation and interpretation, LLMs increasingly serve as the reasoning core of agentic systems, enabling software to reason about tasks and act toward goals (Yao et al., 2022; Guo et al., 2024). This marks a shift from conventional software governed by predefined execution logic to systems dynamically guided by language-model reasoning (Braiek & Khomh, 2020).

At the center of such systems is the *agentic harness*, which connects the model to its environment by constructing its context from user queries, prior observations, and developer-defined directives, and translating model outputs into actions such as accessing the web, processing documents, and executing code (Yang et al., 2024; Yao et al., 2022; Schick et al., 2023; Wang et al., 2025).

Unlike user instructions or prior observations, developer-defined directives persist across interactions and often encode domain-specific or proprietary knowledge absent from pretraining data. For example, a company may maintain customer-specific advertising guidelines that it cannot share with a closed-source model provider. Fine-tuning or maintaining an in-house model presents an alternative, but can be costly when such directives evolve with customer or seasonal demands. Skill files instead allow this knowledge to be updated at inference time, independently of the underlying model, while providing a consolidated structure for maintaining such information (Li et al., 2026).

Skill files can themselves introduce failures into agentic systems. These files are authored by developers to encode domain-specific knowledge or generated by LLMs from user interactions in self-evolving systems (Ouyang et al., 2026). As a result, they can contain inconsistencies and errors similar to those found in software specifications and configuration files (Nagaraja et al., 2004; Enck et al., 2009; Xu et al., 2016; Klischies et al., 2023). Incorrect (Xu et al., 2016), contradictory (Rahman et al., 2024), ambiguous (Klischies et al., 2023), or contextually inappropriate directives can misguide an agent, leading to task failure or poor user experience. This problem is compounded because LLM-based agents are optimized to follow provided instructions (Bianchi et al., 2024; Sun et al., 2020), yet may lack the knowledge or external evidence needed to recognize faulty instruc-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tions (Zhang et al., 2025b). This motivates **Skill-level root cause analysis (RCA)**: identifying, from agent executions and their outcomes, the Skill directives likely responsible for observed failures.

Failures in agentic systems can broadly be characterized as *hard* or *soft* (Zhang et al., 2026; Avizienis et al., 2004). Hard failures prevent task completion due to errors, or service refusals, whereas soft failures occur when a task completes but the output is unsatisfactory to the user or an external evaluator. While prior work (Zhang et al., 2025a; Cemri et al., 2025) has largely focused on hard failures, we study three soft failures arising from Skill files. *Wrong content* refers to incorrect or outdated directives; *contradiction* occurs when directives prescribe incompatible behaviors for the same capability (Hausmann et al., 2002); and *ambiguity* arises when a directive admits multiple plausible interpretations and the agent follows an unintended one (Ezzini et al., 2021).

Although these failure modes affect agent behavior differently, they share a common diagnostic challenge: identifying the Skill directive responsible for a failure. This is difficult for three reasons. First, including a Skill file in the model context does not reveal which directive influenced an action. Unlike traditional software, where execution coverage indicates which components were exercised (Jones et al., 2002), agentic systems lack instruction-level runtime signals; actions may instead depend on task instructions, prior observations, or the model's own reasoning. Second, relevant information may be distributed across the Skill file, with the same rule reinforced through procedures or examples, so localizing a single occurrence may miss related or conflicting directives. Third, proprietary directives may encode knowledge absent from the model's training data, making their correctness difficult to assess from a single failed execution. Comparing successful and failed traces can provide stronger evidence, but directly supplying large collections of logs to an LLM incurs substantial token overhead (Hooper et al., 2025). Since execution traces are a primary source of evidence for post-deployment debugging, they should be enriched with information that improves debuggability. Traditional software commonly exposes such runtime information through verbose logs (Zhu et al., 2015; Xu et al., 2016); however, modern agentic harnesses are primarily optimized for task execution and user interaction, with limited diagnostic support.

To address these challenges, we introduce SKILLROOT, an attribution-based fault localization framework that traces failures to Skill directives in four stages. First, we augment agentic harnesses with action-level attribution, assigning each action to four context sources: task instruction, Skill, prior observation, or own judgment. Second, we construct a Skill Knowledge Graph that represents directives, entities, and their relationships, consolidating attribution evidence across related content. Third, inspired by spectrum-based fault localization (Jones et al., 2002; Abreu et al., 2007), we combine attribution and graph evidence to compute a suspiciousness prior that ranks directives by their association with failed executions. Finally, for a new failure, an LLM-based RCA judge combines this prior with the execution trace to identify the responsible Skill chunks.

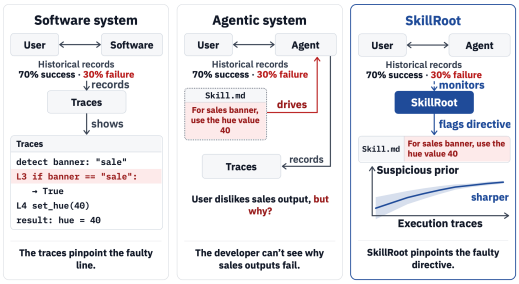

Figure 1: SKILLROOT monitors user–agent interactions and associates each Skill directive with a suspiciousness prior based on its association with failures. As execution traces accumulate, this prior becomes more informative for identifying directives likely responsible for observed failures.

Figure 1 illustrates how, in traditional software systems, RCA can identify problematic control decisions through detailed logging of control-flow paths. In agentic harnesses, however, this is more difficult because existing logs do not explicitly capture the context or control decisions that lead the agent to a particular action. SKILLROOT addresses this gap by augmenting the harness to record such context in execution traces, enabling failures to be associated with specific directives provided to the model. It then monitors user–agent interactions and updates the suspiciousness of Skill directives as new evidence accumulates, helping developers pinpoint directives requiring correction or refinement, and improving system reliability over time. Our main contributions are as follows:

- We introduce an action-level attribution scheme, where an agent attributes its action to the specific context source that influenced its decision.

- We propose a Skill Knowledge Graph with an explicit ontology to organize Skill instructions and capture semantic relationships among directives distributed across the Skill file.
- We develop an attribution-based fault localization framework that aggregates directive activations across successful and failed executions to compute a suspiciousness prior over Skill directives. This prior guides root cause localization for new failures, improving performance by up to 6% over standard RCA baselines.

2 Related Work

Failure analysis and Agentic system Despite recent advances, agentic systems still fail after deployment (Cemri et al., 2025; Deshpande et al., 2025). These failures are recorded in execution traces, which developers inspect to identify possible root causes (Zhang et al., 2025a; Chen et al., 2026; Ma et al., 2026). In traditional software, similar issues can often be traced to specific configurations by comparing successful and failed executions (Liblit et al., 2005; Attariyan et al., 2012; Xu et al., 2016; Sun et al., 2020). In agentic systems, this is harder because the influence of individual Skill directives on actions is not directly observable, while analyzing large trace collections with closed-source models incurs substantial token cost. Recent work instead localizes failures within individual traces by identifying the first failing step (Zhang et al., 2025a), training models for failure analysis (Kong et al., 2026), or finding the correction step that restores task success (Ma et al., 2026). However, such analyses are execution-specific and may not prevent recurring failures. In this work, we focus on persistent Skill directives shared across executions, by aggregating evidence from successful and failed traces we identify directives consistently associated with failure and provide actionable targets for improving reliability.

Failure Attribution Attribution methods have long been used to trace model behavior back to influential training examples, model components, or input context (Koh & Liang, 2017; Ghorbani & Zou, 2019; Dai et al., 2022; Cohen-Wang et al., 2024). Many of these methods, however, rely on gradients, activations, or repeated model evaluations (Phukan et al., 2024; Koh & Liang, 2017; Cohen-Wang et al., 2024), which makes them difficult to use with frontier models accessed through APIs. In agentic systems, recent work has therefore focused more on localizing failures within execution traces (Zhang et al., 2025a; Kong et al., 2026) or using external judges and guardrails to assess whether actions are supported by the available context (Radevski et al., 2025; She et al., 2026). We take a different approach and capture attribution as the agent executes: for each action, the agent records the context source it relied on and, when the action is Skill-based, the corresponding Skill evidence. This creates attribution signals that can be aggregated across executions for failure localization, similar in spirit to diagnostic logging in traditional software (Zhu et al., 2015).

Skill Knowledge Graph Skill files are increasingly used to provide closed source models with task specific guidance and capabilities that may not be captured during training (Hong et al., 2026; Li et al., 2026). A single Skill can contain requirements, procedures, examples, and constraints, often expressing the same directive in different forms or sections (Hong et al., 2026). This makes it difficult to reason about individual Skill instruction in isolation. Knowledge graphs provide a natural way to capture these relationships by organizing entities and their connections under a predefined ontology (Preece et al., 2006; Gandon et al., 2017; Hogan et al., 2021; Fang et al., 2024; Zhang et al., 2023). While prior work on skills has focused mainly on retrieval or relationships across skills (Cho et al., 2026; Liu et al., 2026), we focus on the structure within a Skill itself. We define a Skill specific ontology and construct a knowledge graph that links related entities, directives, procedures, and examples, allowing the full context of an instruction to be recovered.

3 Method

3.1 Problem Formulation

Consider an agentic system $\mathcal{A}$ deployed in a real-world environment, whose behavior is guided by **Skill files** containing a set of directives $\mathcal{S} = \{d_1, d_2, \ldots, d_n\}$. After deployment, some of these directives may contribute to failures across tasks. We refer to the subset responsible for such failures as the **culprit set**, $\mathcal{C}_s = \{d_1^c, \ldots, d_k^c\}$, where $\mathcal{C}_s \subseteq \mathcal{S}$.

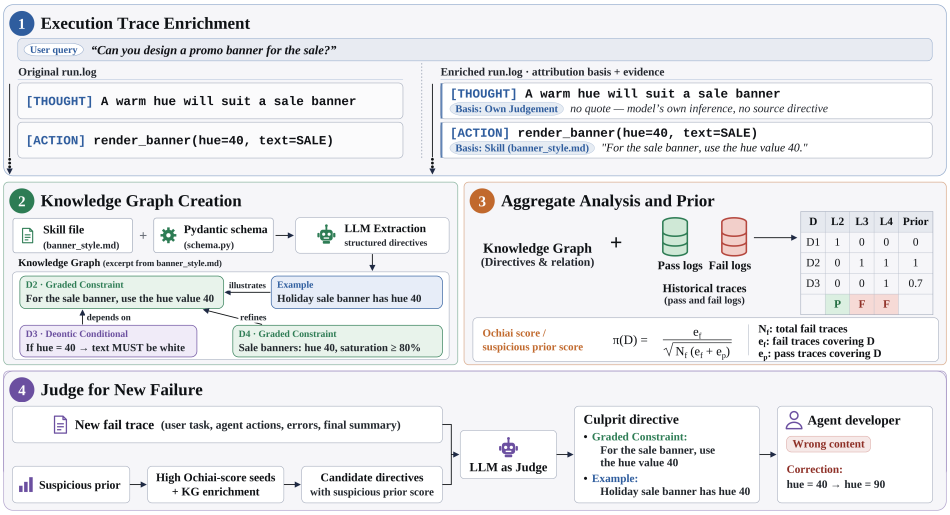

Figure 2: Architecture of our failure localization framework. (1) Attribution records the context supporting each agent action; (2) the Skill Knowledge Graph links related directives and entities; (3) aggregate analysis computes an *suspiciousness prior* using historical executions; and (4) LLM-based judge combines this prior with a failure trace to localize responsible Skill directives.

Given a set of historical execution traces $\mathcal{L} = \{l_1, \ldots, l_m\}$ generated by $\mathcal{A}$, where each trace l_i is associated with an outcome $y(l_i) \in \{0, 1\}$, with 0 denoting failure and 1 success, our goal is to use these executions to estimate a prior $\pi_f(d)$ over directives $d \in \mathcal{S}$ that captures their association with failure. For a new failed execution l_{new}, we combine this prior with new evidence to localize the directives responsible for the failure and recover the corresponding culprit set $\mathcal{C}_s$.

3.2 Approach

Our approach localizes the Skill directives responsible for failure through four stages:

- **Execution Trace Enrichment**: We first augment the harness to generate an enriched execution trace that records not only which context source, such as the Skill, task instruction, prior observation, or own judgment, drives an agent's action, but also the specific supporting text within that source that influenced it.
- **Skill Knowledge Graph**: Since related information may appear across a Skill file, we organize its directives and entities as a knowledge graph, linking attribution evidence to related or dependent instructions.
- **Aggregate Analysis**: Using this information, we identify which directives are active in each trace and compare their occurrence across failed and successful executions. An Ochiai-based suspiciousness score $\pi_f(d_i)$ then summarizes how strongly each directive $d_i \in \mathcal{S}$ is associated with failure, yielding a prior for subsequent RCA.
- **Judge Analysis**: Finally, given a new failed execution l_{new}, an LLM-based judge evaluates a candidate set consisting of the most suspicious directives, their related mentions from the KG, and any contradictory directives identified by the KG, together with evidence from the trace, to identify a suspect set $\mathcal{C}_s \subseteq \mathcal{S}$ most likely responsible for the failure.

3.3 Execution Trace Enrichment

To localize a faulty Skill directive, we need to know how it influenced the agent's behavior during execution. This is not directly visible in current agentic harnesses, where task instructions, Skill content, prior observations, and the agent's own reasoning are typically combined into a shared context before the next action is generated. As a result, even when a Skill file is present, a particular action may actually be driven by another source. First step for RCA therefore requires tracing each action back to both the context source that influenced it and the specific text within that source. To

make this provenance explicit, we record it directly in the execution trace. We modify the tool-call schema and use constrained decoding so that the agent is forced to identify the source supporting each action and, when applicable, quotes the relevant context. For each tool call, the agent attributes its action to one of four sources: **Skill**, **Task Instruction**, **Prior Observation**, or **Own Judgment**. For the first three sources, the trace also records the supporting text and a counterfactual describing how the action would change without it. This gives us action-level evidence for tracing how Skill directives influence agent behavior. Details of the schema and logging procedure are provided in Appendix E, and grounding accuracy is evaluated in Appendix F.

3.4 Skill Knowledge Graph

While attribution can identify the directive supporting a particular action, related information may appear elsewhere in the Skill file. Directives are often repeated, refined, or constrained across rules, examples, procedures, and *"Do/Don't"* sections, so analyzing a single occurrence may miss related or conflicting information. To capture these relationships, we represent each Skill file as a Knowledge Graph (KG) with typed *statement* and *entity* nodes connected by semantic relations.

We construct the KG in two stages. First, a structured LLM call segments the Skill file into atomic statements, assigns each statement a semantic type, extracts type specific attributes from a predefined ontology, and identifies the associated entities. The ontology includes statement types such as **DeonticConditional** for conditional requirements or prohibitions (e.g., *"if condition X holds, perform action Y"*), **GradedConstraint** for measurable constraints (e.g.,*"use at most three retries"*), **Procedure** for ordered execution steps, and **Example** for illustrative instances, along with types for preferences, definitions, capabilities, scopes, facts, and meta policies. Entity nodes capture concrete elements such as API calls, arguments, resources, conditions, actions, and terms. Since these entities are linked to individual directives, their occurrence in runtime traces can be combined with attribution information to determine when a directive is active during execution.

Second, we connect related statements through deterministic and semantic relations. Explicit references, procedural order, and shared-entity links are constructed using lexical matching, while relations such as *depends-on*, *refines*, *scopes*, *illustrates*, *overrides*, and semantic contradictions are identified using a separate LLM call over the Skill text and extracted statements. The resulting graph consolidates related mentions of a directive, allowing downstream analysis to reason over the Skill as a whole rather than isolated text spans. Further details of the KG and ontology are provided in Appendix D.

3.5 Aggregate Analysis

Together, the KG and attribution signals link agent actions to Skill directives while consolidating related occurrences across the file. To isolate faulty directives, we then examine which directives are most strongly associated with failure across historical successful and failed traces. This requires determining when each directive is active in a trace. We use two complementary signals for this: the agent's attribution identifies directives that directly influence its actions, while KG-linked entities are matched against the agent's actions, errors, and final execution summary to capture additional evidence of their use. For conditional statements, we additionally apply an NLI-based trigger check and mark the directive active only when its condition is satisfied (Bostrom et al., 2022; Chen et al., 2021). We use these activation signals to compare how often each directive appears in successful and failed executions, and define an incidence matrix $\mathcal{I}$ where $\mathcal{I}(d, l_i) = 1$ if directive $d \in \mathcal{S}$ is active in trace l_i, and 0 otherwise.

3.5.1 Failure-association based prior

Once directive activations are aggregated across executions, we quantify their association with failure. A directive should receive a high suspiciousness score when it occurs frequently in failed executions but less often in successful ones. We use the Ochiai coefficient for this purpose. Ochiai is closely related to cosine similarity and has been widely used to measure normalized association between binary occurrence patterns in ecology, including in software fault localization (Ochiai, 1957; Abreu et al., 2007; Zhou & Leydesdorff, 2016). In our setting, we apply Ochiai to the activation pattern of each Skill directive across executions and compare it with the binary failure pattern. A high score therefore indicates that the executions in which a directive is active overlap strongly with the

executions that fail. For directive d_j, we define $\pi(d_j) = \boldsymbol{e}_{\text{fail}}(d_j)/\sqrt{\mathcal{N}_{\text{fail}}\left[\boldsymbol{e}_{\text{fail}}(d_j) + \boldsymbol{e}_{\text{pass}}(d_j)\right]}$ where, $\boldsymbol{e}_{\text{fail}}(d_j)$ and $\boldsymbol{e}_{\text{pass}}(d_j)$ denote the number of failed and successful executions in which d_j is active and $\mathcal{N}_{\text{fail}}$ is the total number of failed executions. The resulting score $\pi(d_j)$ is an ochiai based suspiciousness prior that captures the association of a single instruction with failure.

3.5.2 Co-occurrence based Failure

Although individual directives can provide strong failure signals, some failures arise only through their interaction. For example, one directive may require a white background for an ad campaign, while another specifies white text. Each may be valid in isolation, but together they can make the text unreadable and cause failure, as illustrated in Figure 3. The suspiciousness prior should therefore capture not only the contribution of individual directives but also their joint association with failure. However, exhaustively evaluating all possible subsets of Skill statements is combinatorially expensive and quickly becomes intractable as the number of directives grows. We therefore use a KG-guided *beam-search* to efficiently explore combinations of related directives that may jointly contribute to failure.

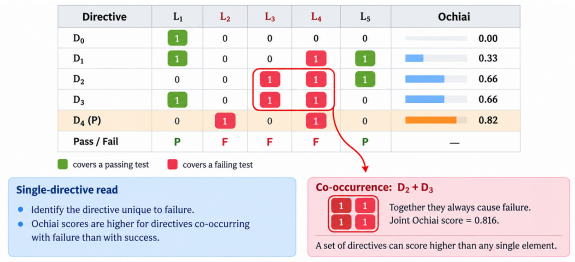

Figure 3: Analysis of single directive and multiple directive failure using the Incidence matrix ($\mathcal{I}$).

Starting from a seed directive that is strongly associated with failure, we iteratively expand the candidate set ($\mathcal{C}$) using neighboring directives in the Skill KG. At each step, a directive is considered for expansion only if it is connected to at least one directive in the current set through an explicit KG edge. Relations such as *refines*, *procedure*, and other semantic links capture dependencies among directives that may jointly contribute to failure, thereby restricting the search to structurally related combinations rather than all possible subsets. Among the valid expansions, we retain only the top B candidate sets according to their set-level suspiciousness score, where B denotes the beam width.

For a candidate set $\mathcal{C}$, we define its activation $\mathcal{I}(\mathcal{C}, l)$ as the intersection of the activation signals of its constituent directives (Section 3.5). Thus, $\mathcal{C}$ is active in a trace only when all of its directives are active. We then count this joint activation across failed and successful executions, denoted by $\boldsymbol{e}_{\text{fail}}(\mathcal{C})$ and $\boldsymbol{e}_{\text{pass}}(\mathcal{C})$, respectively, and use these counts to compute the set-level Ochiai score $\pi_c(\mathcal{C})$:

$$\mathcal{I}(\mathcal{C}, l) = \prod_{d_j \in \mathcal{C}} \mathcal{I}(d_j, l), \qquad \boldsymbol{e}_{\text{fail}}(\mathcal{C}) = \sum_{i:\, y(l_i)=0} \mathcal{I}(\mathcal{C}, l_i), \qquad \boldsymbol{e}_{\text{pass}}(\mathcal{C}) = \sum_{i:\, y(l_i)=1} \mathcal{I}(\mathcal{C}, l_i),$$
$$\pi_c(\mathcal{C}) = \frac{\boldsymbol{e}_{\text{fail}}(\mathcal{C})}{\sqrt{\mathcal{N}_{\text{fail}}\left[\boldsymbol{e}_{\text{fail}}(\mathcal{C}) + \boldsymbol{e}_{\text{pass}}(\mathcal{C})\right]}}. \tag{1}$$

We repeat this search from different seed directives to recover failure patterns not captured by earlier groups. Seeds are prioritized by coverage of previously unexplained failures, followed by overall failure coverage and individual Ochiai score. This favors groups that are both strongly associated with failure and complementary to those already identified. Once the expansion terminates, the suspiciousness of each directive is updated using the strongest candidate set in which it appears:

$$\pi_f(d_j) = \max\{\pi(d_j),\ \pi_{\text{group}}(d_j)\} \text{ where, } \pi_{\text{group}}(d_j) = \max_{\mathcal{C}:\, d_j \in \mathcal{C}} \pi_c(\mathcal{C}), \tag{2}$$

Here, $\pi_{\text{group}}(d_j)$ captures the strongest failure association among discovered candidate sets containing d_j. The final score π_f preserves the directive's individual suspiciousness while allowing it to increase when its co-occurrence with related directives is more strongly associated with failure. Further details on the co-occurrence strategy and prior construction are provided in Appendix G.

3.6 Judge Analysis

The suspiciousness prior captures which Skill statements are repeatedly associated with failure across historical executions, but a high prior alone does not establish that a given directive caused

the specific failure under analysis. We therefore use an LLM-based judge as the final RCA stage to identify culprit directives from a candidate set. This set starts with the highest-scoring statements and is expanded using the Skill KG: neighboring statements are added to recover semantically related directives with lower prior scores, while both endpoints of contradiction edges are included for analysis.

For a new failed execution l_{new}, the judge receives the candidate set and associated suspiciousness scores π_f, together with details of the new failure trace l_{new}. It then uses the task context and observed agent behavior to determine which directives best explain the failure. Thus, historical executions provide statistical evidence to narrow the search, while the current failure trace provides the context required for final root-cause attribution. Additional details on the judge prompt and trace analysis are provided in Appendix I.

4 EXPERIMENTS

4.1 SETTING

Setup : We conduct experiments with OpenHands (Wang et al., 2025; 2026), an open-source multi-domain agent harness used in prior work on software engineering (Pan et al., 2025) and scientific discovery (Chen et al., 2025). We use Sonnet-4.6 for agent execution because of its lower inference cost, strong performance across diverse agentic tasks (Kapoor et al., 2026; Chen et al., 2025; Thomas, 2026), and evidence of similar failure behaviors across model families (Thomas, 2026; Mosquera-Gómez et al., 2026). Additionally, GPT-5.5 and Opus-4.6 are used as RCA judges.

Data: For our analysis, we use Skill files from SkillBench (Li et al., 2026), the open-source Claude Skills repository, and synthetic Skills. For each Skill, we generate around 10-13 user queries and deterministic binary verifiers following Li et al. (2026) to verify the associated result; all tasks succeed with the correct Skill file. We then edit the Skill files according to our three error categories, producing successful and failed executions within each Skill family. In total, we analyze 17 Skill families and 530 execution traces across tasks to identify the edits responsible for failure (see Appx C) .

Baseline : We compare four Skill-based RCA strategies. **1. Who and When** adapts Zhang et al. (2025a) to identify the failure-inducing step using Skill attribution recorded during execution. Inspired by Lian et al. (2025), **2. Skill Audit** asks an LLM judge to inspect the Skill file for faulty directives, while **3. Log and Skill Audit** additionally provides the failed execution trace to contextualize observed agent behavior.

We evaluate SKILLROOT using two Skill representations: **SKILLROOT (KG)** directly evaluates directives extracted from the Skill Knowledge Graph, whereas **SKILLROOT (paragraph)** provides broader paragraph-level context, assigning each paragraph the highest Ochiai score among its directives. This comparison tests whether broader context improves the judge's reasoning for RCA.

Analysis : We evaluate the system along three dimensions. First, we study agent behavior under *wrong content*, *contradiction*, and *ambiguity* in Skill files, examining how attribution differs between successful and failed executions and whether agents can override faulty directives when sufficient context is available. Second, we compare RCA strategies on their ability to localize failure-inducing Skill directives. Third, we ablate attribution information associated with log enrichment to measure its contribution to culprit-directive localization.

Metrics: Our evaluation data contains ground truth, because each task succeeds with the original Skill file and fails only after an injected edit, providing the known culprit directive. We test whether this directive can be localized from the modified Skill files and success and failure traces. We report precision, recall, and F1 against the injected edit, averaged over K-fold cross-validation, with one failed trace held out for the RCA judge in each fold. The remaining traces are used to construct the suspiciousness prior. Further details on the dataset and evaluation are provided in Appendix C.

4.2 FAILURE AND ATTRIBUTION ANALYSIS

Figure 4 summarizes task success across error types and the corresponding action-attribution distribution. As shown in panel (a), ambiguity achieves the highest success rate (82.6%), followed by contradiction (71.2%) and wrong content (66.7%). This pattern reflects the contextual evidence

(a) Task success by error type

Error type	Success (%)
Ambiguity	82.6
Contradiction	71.2
Wrong content	66.7

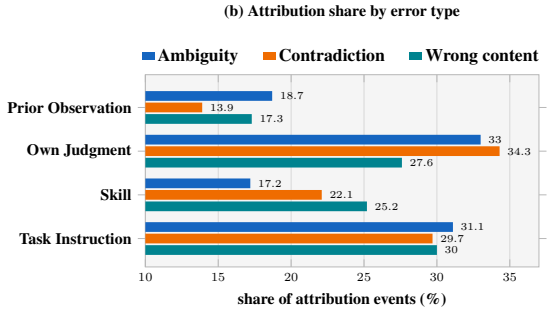

Figure 4: Task outcomes and action-attribution patterns across three injected error types. **(a)** Task success rates for three error type. **(b)** Action-attribution distribution across different sources.

available for detecting Skill errors: ambiguity and contradiction often expose inconsistencies, such as conflicting rules or mismatches between vague instructions and specific examples, encouraging the agent to rely on alternative context sources. In contrast, an incorrect value or rule may remain locally plausible without relevant proprietary or external knowledge. Panel (b) supports this interpretation: Skill attribution is lower for ambiguity (17.2%) and contradiction (22.1%) than for wrong content (25.1%), suggesting greater reliance on alternative context for the former two. Wrong-content errors provide less contextual evidence about whether a directive is correct, making them harder to detect and resolve.

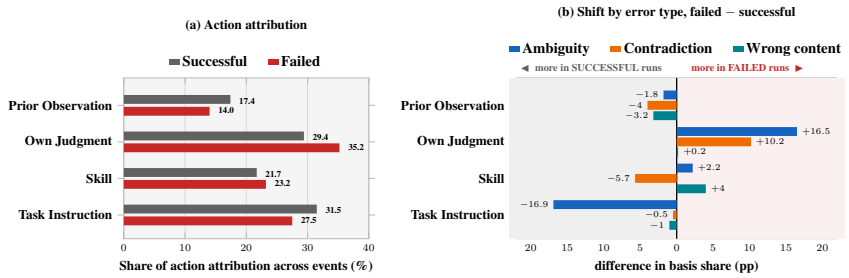

Figure 5: Action-attribution patterns in successful and failed runs. **(a)** Overall attribution distribution. **(b)** Error-type-specific percentage-point change from successful to failed runs; positive values indicate greater reliance in failures, and negative values greater reliance in successes.

Figure 5 compares reliance on different context sources between successful and failed executions. Across error types, panel (a) shows that failed runs rely more on faulty Skill instructions and the agent's own judgment, and less on task instructions and prior observations. This suggests that failures are associated with following faulty directives or underusing contextual evidence that could help resolve them. However, these effects vary across error types.

Panel (b) shows that different failure modes are resolved using different sources of context. Wrong-content failures remain closely tied to the Skill itself, since the agent may simply follow a directive containing an incorrect value or rule. Ambiguity and contradiction behave differently: failed runs rely more heavily on the agent's own judgment, by 16.5 and 10.2 percentage points, respectively. In successful runs, however, the agent makes greater use of context that can help resolve the issue. Skill attribution is 5.7 points higher for contradiction, whereas task-instruction attribution is 16.9 points higher for ambiguity. Together, these results suggest that ambiguity is often resolved by clearer task information and contradiction by consulting related Skill content, while wrong content remains harder when the error originates in the Skill itself.

4.3 Comparison with baselines

Table 1 summarizes the performance of our prior-based RCA method. The largest gains appear for *wrong content*: KG-based chunking achieves an F_1 of 0.842 with Opus-4.6, while paragraph-

based chunking reaches 0.879 with GPT-5.5,improving over Log and Skill Audit by 3.9 and 6.6 points, respectively. In contrast, Skill Audit performs suboptimally in this setting, with F_1 scores of 0.218 and 0.201. This reflects the difficulty of wrong-content errors, where an incorrect value may appear plausible in isolation and a single failed trace may not provide enough evidence for localization. Aggregating evidence across executions reveal directives that occur disproportionately in failed runs, giving the RCA judge a stronger signal for localization.

Table 1: Performance for localizing the edit responsible for each failure. Within each model block, **bold** denotes the best value per column and underline the second best. Results report mean recall (R), precision (P), and F_1 across K-fold validation.

		Wrong content			Contradiction			Ambiguity		
	Method	R	P	F_1	R	P	F_1	R	P	F_1
OPUS-4.6	Who and When	0.374	0.549	0.444	0.413	0.461	0.435	0.174	0.470	0.254
	Skill Audit	0.199	0.243	0.218	0.850	0.583	0.692	0.333	0.458	0.386
	Log and Skill Audit	0.755	**0.858**	0.803	**0.930**	0.750	0.829	**0.631**	**0.752**	**0.686**
	SKILLROOT *(paragraph)*	0.806	0.844	0.824	0.885	**0.848**	**0.864**	**0.631**	**0.752**	**0.686**
	SKILLROOT *(KG)*	**0.828**	0.856	**0.842**	0.890	0.814	0.848	0.452	**0.752**	0.560
GPT-5.5	Who and When	0.360	0.501	0.418	0.413	0.461	0.436	0.413	0.461	0.436
	Skill Audit	0.212	0.192	0.201	0.750	0.534	0.622	0.313	0.417	0.357
	Log and Skill Audit	0.789	0.839	0.813	**0.840**	**0.882**	**0.860**	0.427	0.729	0.532
	SKILLROOT *(paragraph)*	**0.855**	**0.907**	**0.879**	0.818	0.878	0.846	0.436	0.648	0.514
	SKILLROOT *(KG)*	0.800	0.860	0.828	0.797	0.868	0.831	**0.457**	**0.737**	**0.548**

Contradiction and ambiguity show a different pattern. For contradiction, relevant evidence is already available in the Skill and execution trace, allowing Log and Skill Audit to remain competitive: it reaches an F_1 of 0.860 with GPT-5.5 and trails our method by 3.5 points with Opus-4.6. Skill Audit achieves high recall but lower precision, suggesting that some apparent conflicts in the Skill are not responsible for the observed failure. Ambiguity is harder to localize because the agent often relies on its own judgment or the task instruction, leaving weaker evidence about the faulty Skill directive. Consequently, recall remains lower than precision across methods and models.

4.4 INFLUENCE OF SKILL ATTRIBUTION

Given the dependence of wrong-content failures on Skill instructions, we evaluate whether attribution improves RCA beyond entity-based matching. As shown in Figure 6, attribution improves Opus-4.6 recall/precision/F_1 from 0.763/0.791/0.776 to 0.828/0.856/0.842. For GPT-5.5, recall remains unchanged, while precision increases from 0.844 to 0.860. This shows that action-level attribution provides evidence for refining Ochiai-based localization of wrong-content errors.

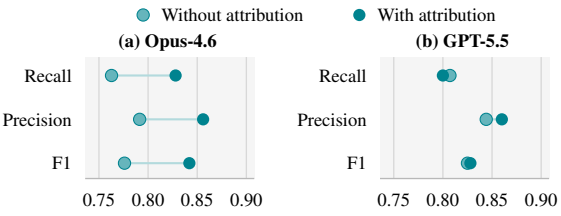

Figure 6: Effect of runtime skill attribution on wrong-content localization.

5 CONCLUSION

We introduced a framework for attributing failures to directives in Skill files of agentic systems. The framework combines action-level attribution, a Skill Knowledge Graph, and activation-based Ochiai analysis to aggregate successful and failed executions into a suspiciousness prior, which an LLM-based judge uses to identify faulty directives. Across error types, aggregate evidence is particularly useful for wrong-content failures, where neither the Skill file nor a single trace may provide sufficient evidence to identify an incorrect directive. As a future work, we plan to improve the attribution technique by leveraging model's attention scores for open source agents.

ETHICS STATEMENT

Our work focuses on identifying errors in Skill files that may contribute to failures in agentic systems, using attribution information and execution traces for analysis. The experiments do not rely on private user data or involve analysis that violates user privacy. We therefore do not identify any direct ethical or privacy concerns associated with the evaluation. More broadly, the proposed methodology is intended to support debugging and improve the reliability of agentic systems by helping developers identify problematic Skill instructions.

REPRODUCIBILITY STATEMENT

We include the key details required to reproduce our study throughout the appendix. Appendix C describes the dataset construction and experimental setup, Appendix E documents the attribution mechanism and associated schema changes, and Appendix I provides the prompts used by the RCA judge. Any remaining details will be released upon acceptance of the work.

REFERENCES

Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, pp. 89–98. IEEE, 2007.

Mona Attariyan, Michael Chow, and Jason Flinn. X-ray: Automating {Root-Cause} diagnosis of performance anomalies in production software. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pp. 307–320, 2012.

Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1 (1):11–33, 2004.

Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Röttger, Dan Jurafsky, Tatsunori Hashimoto, and James Y Zou. Safety-tuned llamas: Lessons from improving the safety of large language models that follow instructions. In *International Conference on Learning Representations*, volume 2024, pp. 34196–34216, 2024.

Kaj Bostrom, Zayne Sprague, Swarat Chaudhuri, and Greg Durrett. Natural language deduction through search over statement compositions. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 4871–4883, 2022.

Houssem Ben Braiek and Foutse Khomh. On testing machine learning programs. *Journal of Systems and Software*, 164:110542, 2020.

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? *Advances in Neural Information Processing Systems*, 38, 2025.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, 2026.

Zeming Chen, Qiyue Gao, and Lawrence S Moss. Neurallog: Natural language inference with joint neural and logical reasoning. In *Proceedings of* SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pp. 78–88, 2021.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, volume 2025, pp. 96934–96990, 2025.

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. Skillret: A large-scale benchmark for skill retrieval in llm agents. *arXiv preprint arXiv:2605.05726*, 2026.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Mądry. Contextcite: Attributing model generation to context. *Advances in Neural Information Processing Systems*, 37:95764–95807, 2024.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

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

William Enck, Thomas Moyer, Patrick McDaniel, Subhabrata Sen, Panagiotis Sebos, Sylke Spoerel, Albert Greenberg, Yu-Wei Eric Sung, Sanjay Rao, and William Aiello. Configuration management at massive scale: System design and experience. *IEEE Journal on Selected Areas in Communications*, 27(3):323–335, 2009.

Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C Briand. Using domain-specific corpora for improved handling of ambiguity in requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1485–1497. IEEE, 2021.

Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. Reano: Optimising retrieval-augmented reader models through knowledge graph generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2094–2112, 2024.

Fabien Gandon, Guido Governatori, and Serena Villata. Normative requirements as linked data. In *JURIX 2017-The 30th international conference on Legal Knowledge and Information Systems*, pp. 1–10, 2017.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pp. 2242–2251. PMLR, 2019.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Jan Hendrik Hausmann, Reiko Heckel, and Gabi Taentzer. Detection of conflicting functional requirements in a use case-driven approach: a static analysis technique based on graph transformation. In *Proceedings of the 24th international conference on software engineering*, pp. 105–115, 2002.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37, 2021.

David Boram Hong, Aaron Imani, and Iftekhar Ahmed. From anatomy to smells: An empirical study of skill. md in agent skills. *arXiv preprint arXiv:2607.01456*, 2026.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32631–32652, 2025.

James A Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pp. 467–477, 2002.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026.

Daniel Klischies, Moritz Schloegel, Tobias Scharnowski, Mikhail Bogodukhov, David Rupprecht, and Veelasha Moonsamy. Instructions unclear: undefined behaviour in cellular network specifications. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3475–3492, 2023.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. Pmlr, 2017.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 52994–53028, 2026.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Xinyu Lian, Yinfang Chen, Runxiang Cheng, Jie Huang, Parth Thakkar, Minjia Zhang, and Tianyin Xu. Large language models as configuration validators. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1704–1716. IEEE, 2025.

Ben Liblit, Mayur Naik, Alice X Zheng, Alex Aiken, and Michael I Jordan. Scalable statistical bug isolation. *Acm Sigplan Notices*, 40(6):15–26, 2005.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 28059–28087, 2026.

Rafael Mosquera-Gómez, Juan Felipe Rodriguez, Martin Diaz Velez, Ivan Alvarenga de Sousa Junior, and Juan Jaramillo. Constrained wikigame: Benchmarking deductive reasoning for multi-step planning. In *ICLR 2026 Workshop on Logical Reasoning of Large Language Models*, 2026.

Kiran Nagaraja, Fábio Oliveira, Ricardo Bianchini, Richard P Martin, and Thu D Nguyen. Understanding and dealing with operator mistakes in internet services. In *OSDI*, volume 4, pp. 61–76, 2004.

Akira Ochiai. Zoogeographic studies on the soleoid fishes found in japan and its neighbouring regions. *Bulletin of Japanese Society of Scientific Fisheries*, 22:526–530, 1957.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *URL https://arxiv. org/abs/2605.06614*, 2026.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 13–19 Jul 2025. URL `https://proceedings.mlr.press/v267/pan25g.html`.

Anirudh Phukan, Shwetha Somasundaram, Apoorv Saxena, Koustava Goswami, and Balaji Vasan Srinivasan. Peering into the mind of language models: An approach for attribution in contextual question answering. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11481–11495, 2024.

Alun Preece, Stuart Chalmers, Craig McKenzie, Jeff Z Pan, and Peter Gray. A semantic web approach to handling soft constraints in virtual organisations. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, pp. 151–161, 2006.

Gorjan Radevski, Kiril Gashteovski, Shahbaz Syed, Christopher Malon, Sebastien Nicolas, Chia-Chien Hung, Timo Sztyler, Verena Heußer, Wiem Ben Rim, Masafumi Enomoto, et al. On synthesizing data for context attribution in question answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16929–16950, 2025.

Mirza Masfiqur Rahman, Imtiaz Karim, and Elisa Bertino. {CellularLint}: A systematic approach to identify inconsistent behavior in cellular network specifications. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 5215–5232, 2024.

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

Yining She, Yiliang Liang, and Eunsuk Kang. Safeguarding llm agents from misalignment through provenance analysis. *arXiv preprint arXiv:2607.01236*, 2026.

Xudong Sun, Runxiang Cheng, Jianyan Chen, Elaine Ang, Owolabi Legunsen, and Tianyin Xu. Testing configuration changes in context to prevent production failures. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 735–751, 2020.

Noel Thomas. Chaosbench-logic v2: Evaluating llm logical reasoning over dynamical systems at scale. *arXiv preprint arXiv:2605.24305*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang H Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, et al. The openhands software agent sdk: A composable and extensible foundation for production agents. *Proceedings of Machine Learning and Systems*, 8:1843–1855, 2026.

Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 619–634, 2016.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Ruoyu Zhang, Yanzeng Li, and Lei Zou. A novel table-to-graph generation approach for document-level joint entity and relation extraction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10853–10865, 2023.

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*, 2025a.

Wentao Zhang, Yan Zhuang, ZhuHang Zheng, Mingfei Zhang, Jiawen Deng, and Fuji Ren. Beyond explicit refusals: Soft-failure attacks on retrieval-augmented generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30281–30302, 2026.

Zhihan Zhang, Shiyang Li, Zixuan Zhang, Xin Liu, Haoming Jiang, Xianfeng Tang, Yifan Gao, Zheng Li, Haodong Wang, Zhaoxuan Tan, et al. Iheval: Evaluating language models on following the instruction hierarchy. 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)*, pp. 8374–8398, 2025b.

Qiuju Zhou and Loet Leydesdorff. The normalization of occurrence and c o-occurrence matrices in bibliometrics using cosine similarities and o chiai coefficients. *Journal of the Association for Information Science and Technology*, 67(11):2805–2814, 2016.

Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. Learning to log: Helping developers make informed logging decisions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pp. 415–425. IEEE, 2015.

APPENDIX

CONTENTS

A Algorithm **16**

B Large Language Model Usage **17**

C Dataset Construction and Evaluation Setup **17**

C.1 Skill File Collection and Domain Coverage 17

C.2 Task and Failure Variant Construction 18

C.3 Selection of the Evaluation Set 18

C.4 Evaluation Protocol 18

C.5 Edit Analysis 18

D Skill Knowledge Graph Construction **19**

D.1 Statement Extraction and Typing. 19

D.2 Entity Extraction 20

D.3 Source Grounding. 20

D.4 Relation Construction. 20

D.4.1 Edge Representation. 21

E Agent Attribution and Logging **21**

E.1 Attribution Taxonomy. 21

E.2 Causal Attribution. 22

E.3 Action Context 23

F Analysis of Attribution Performance **24**

F.1 Performance of action grounding 24

F.2 Performance of the context attribution 25

G Aggregate Analysis and Suspiciousness Estimation **26**

G.1 Trace Representation 26

G.2 Statement Activation 26

G.3 Ochiai Suspiciousness 27

G.4 KG-Guided Beam Search for Directive Sets 28

G.4.1 Set-level Activation. 28

H Ablation of Co-occurrence Merging for Root Cause Analysis **29**

I LLM Judge Prompt and Failure Trace Analysis **30**

I.1 Failure Trace Analysis 30

I.1.1 System Prompt 30
I.1.2 Multi-Select Instruction 31
I.1.3 Ochiai-Score Guidance 31
I.1.4 Failure-Specific Prompt 32

A ALGORITHM

Our approach consists of four stages: execution trace enrichment, Skill knowledge graph construction, aggregate analysis, and judge analysis. Alg. 1 covers the first stage, and Alg. 2 covers the remaining three.

Execution Trace Enrichment. (§3.3) We execute the agent with Skill file $\mathcal{S}$ on tasks $\mathcal{T}$ using a modified harness $\mathcal{H}$. In addition to standard action and observation records, $\mathcal{H}$ requires the agent to identify whether each action is grounded in the Skill, task instruction, prior observation, or its own judgment. For actions grounded in an explicit context source, the harness also records the supporting context and a counterfactual describing how the action would change without it. Each trace is then labeled as passing or failing, yielding grounded trace sets $\mathcal{L}_{\text{pass}}$ and $\mathcal{L}_{\text{fail}}$. Because this grounding is recorded during execution, the traces can later be analyzed without rerunning the agent.

Skill Knowledge Graph. (§3.4) We segment $\mathcal{S}$ into typed atomic directives and organize them into a knowledge graph $\mathcal{K}$, with edges capturing lexical and semantic relations such as *contradicts*, *depends-on*, and *refines*.

Aggregate Analysis. (§3.5) For each trace, we identify which directives are active. A directive is marked active if the agent explicitly attributes an action to it, if its associated entities appear in the trace, or, for conditional directives, if an NLI model determines that its trigger condition is satisfied. We then compare these activations across passing and failing traces to compute an Ochiai suspiciousness score π for each directive. Because some failures arise from interactions among multiple directives, we further perform a beam search over $\mathcal{K}$ with M restarts to identify directive groups jointly associated with failure and propagate each group score back to its members. The final score π_{final} is the maximum of the individual and group-level scores.

Judge Analysis. (§3.6) We construct a candidate set from the top-c directives, expand it with their h-hop undirected neighbors in $\mathcal{K}$ and all directives involved in contradiction edges, and pass the resulting set to an LLM judge. Using the evidence in l_{new} together with the suspiciousness scores, the judge selects the final suspect set $\mathcal{C}^S$.

Algorithm 1 Execution Trace Enrichment

Require: Skill file $\mathcal{S} = \{d_1, \ldots, d_n\}$; task set $\mathcal{T} = \{t_1, \ldots, t_m\}$; modified harness $\mathcal{H}$
Ensure: Grounded execution traces $\mathcal{L} = \mathcal{L}_{\text{fail}} \cup \mathcal{L}_{\text{pass}}$

$\mathcal{L}_{\text{fail}}, \mathcal{L}_{\text{pass}} \leftarrow \emptyset$
for $t_i \in \mathcal{T}$ **do**
  Execute the agent with $\mathcal{S}$ on t_i under $\mathcal{H}$ to obtain trace l_i
  for each action $a \in l_i$ **do**
    Record its grounding source $g(a) \in \{\text{Skill}, \text{ task instruction}, \text{ prior observation}, \text{ own judgment}\}$
    if $g(a) \neq$ own judgment **then**
      Store the supporting span and its counterfactual in l_i
    end if
  end for
  if l_i failed **then**
    $\mathcal{L}_{\text{fail}} \leftarrow \mathcal{L}_{\text{fail}} \cup \{l_i\}$
  else
    $\mathcal{L}_{\text{pass}} \leftarrow \mathcal{L}_{\text{pass}} \cup \{l_i\}$
  end if
end for
return $\mathcal{L}_{\text{fail}} \cup \mathcal{L}_{\text{pass}}$

Algorithm 2 Skill Directive Localization

Require: Skill file $\mathcal{S} = \{d_1, \ldots, d_n\}$; grounded traces $\mathcal{L}$ from Alg. 1, with $\mathcal{N}_{\text{fail}} = |\mathcal{L}_{\text{fail}}| > 0$; new failed execution l_{new}; beam width B; restarts M; hop limit h; shortlist size c
Ensure: Suspect set $\mathcal{C}^S \subseteq \mathcal{S}$

Skill Knowledge Graph

Segment $\mathcal{S}$ into typed atomic directives with attributes and entities $\mathcal{E}$
Build $\mathcal{K} = (\mathcal{S}, \mathcal{E}, \mathcal{R})$ with lexical and semantic relations (*contradicts*, *depends-on*, *refines*, *references*, ...)

Aggregate Analysis {§3.5}

for $l_i \in \mathcal{L}$, $d_j \in \mathcal{S}$ **do**
  $\eta(d_j, l_i) \leftarrow \mathbf{1}[\,\mathcal{E}(d_j)$ matched in $l_i\,]$ if $\mathcal{E}(d_j) \neq \emptyset$, else 0
  $\tau(d_j, l_i) \leftarrow$ NLITRIGGER(d_j, l_i) if d_j is conditional, else 0
  $\mathcal{I}(d_j, l_i) \leftarrow \mathbf{1}[\,d_j$ attributed in $l_i\,] \vee \eta(d_j, l_i) \vee \tau(d_j, l_i)$
end for
for $d_j \in \mathcal{S}$ **do**
  $\boldsymbol{e}_{\text{fail}}(d_j),\ \boldsymbol{e}_{\text{pass}}(d_j) \leftarrow$ column sums of $\mathcal{I}$ over $\mathcal{L}_{\text{fail}}$ and $\mathcal{L}_{\text{pass}}$
  $\pi(d_j) \leftarrow \dfrac{\boldsymbol{e}_{\text{fail}}(d_j)}{\sqrt{\mathcal{N}_{\text{fail}}\big(\boldsymbol{e}_{\text{fail}}(d_j) + \boldsymbol{e}_{\text{pass}}(d_j)\big)}}$, or 0 if inactive
end for

Co-occurrence Search over Directive Groups

for $s = 1$ **to** M **do**
  Select a seed from the candidate pool, prioritising failures uncovered by earlier restarts
  Grow the seed over edges of $\mathcal{K}$ by beam search (width B), scoring each group by π_c {Eq. equation 1}
  Credit each surviving group's score to its members via π_{group}; mark the failures it covers
end for
$\pi_{\text{final}}(d_j) \leftarrow \max\{\pi(d_j),\ \pi_{\text{group}}(d_j)\}$ for all $d_j \in \mathcal{S}$ {Eq. equation 2}

Judge Analysis {§3.6}

$\mathcal{C}^S \leftarrow$ top-c directives of $\mathcal{S}$ ranked by π_{final}
$\mathcal{C}^S \leftarrow \mathcal{C}^S \cup \{\, d \in \mathcal{S} : d$ is within h undirected hops of $\mathcal{C}^S$ in $\mathcal{K}\,\}$
$\mathcal{C}^S \leftarrow \mathcal{C}^S \cup \{\, d \in \mathcal{S} : d$ is an endpoint of a *contradicts* edge in $\mathcal{R}\,\}$
Extract the user query, action blocks, errors, and final summary from l_{new}
$\mathcal{C}^S \leftarrow$ subset of $\mathcal{C}^S$ selected by the LLM judge given l_{new} and $\{\pi_{\text{final}}(d)\}_{d \in \mathcal{C}^S}$
return $\mathcal{C}^S$

B Large Language Model Usage

Large language models were used to assist with manuscript writing, and grammer check, including coding support through tools such as Claude Code. As described in Appendix C, LLMs were also used to generate Skill-file edits and task queries associated with them. All generated edits and tasks were manually reviewed before inclusion in the experiments. Further details on the models, prompts, and verification procedure are provided in the Appendix.

C Dataset Construction and Evaluation Setup

C.1 Skill File Collection and Domain Coverage

We construct our evaluation set from 17 Skill file families, covering a broad range of application domains and approximately 530 execution traces generated using Sonnet 4.6 with an OpenHands based agentic harness (Wang et al., 2025). The skills are drawn from three sources: publicly available Claude skills[1], SkillBench skills (Li et al., 2026), and synthetic skills designed to emulate organization specific procedures and policies. The synthetic Skills capture settings where the information required for correct agent behavior is proprietary or absent from the model's pretraining data, including internal compensation policies, document-retention requirements, pricing guidelines, and brand-specific production rules.

The complete collection spans eight broad domains: AI assistant behavior, office and administrative workflows, natural sciences, creative production, design and branding, software engineering, media and communications, and cybersecurity.

[1] https://github.com/anthropics/skills

C.2 Task and Failure Variant Construction

Each Skill family consists of a single Skill file and approximately 10–13 task queries, generated with LLM assistance, that require the agent to use that Skill during execution. Following a setup similar to SkillBench (Li et al., 2026), each task is paired with a verifier that checks whether the final agent output satisfies the task requirements.

Starting from the original Skill file, we use LLM assistance followed by human verification to construct variants for the three failure modes studied in this work: *wrong content*, *contradiction*, and *ambiguity*. We modify only the Skill instructions while keeping the user task and execution environment unchanged, so that any resulting failure can be attributed to the Skill-file edit.

For *wrong content*, we modify a factual value, numeric parameter, operator, or convention used by the Skill, such as an aggregation function, comparison operator, or formatting rule, and propagate the modified value consistently across its occurrences. For *contradiction*, we insert a new instruction elsewhere in the Skill that explicitly conflicts with an existing directive, while preserving a natural document structure. For *ambiguity*, we replace a precise instruction, such as a numeric threshold, with a less specific description while retaining a more precise implementation or example elsewhere in the file. The remainder of the Skill and task environment is kept unchanged across variants. The changes were incorporated with the help of an LLM agent, and with human verification.

C.3 Selection of the Evaluation Set

Not every injected modification leads to an observable task failure. Before introducing any modification, we therefore verify that the agent successfully completes the corresponding tasks using the original Skill file. We then execute the modified variants and retain only families for which at least one injected modification results in a failed execution. Further, to ensure that the Skill file drives the behavior for the given family, we do an ablation and prioritize only those families which fails with out Skill file on a given task, ensuring that the specific knowledge is underrepresented in the internal knowledge of the agent. This filtering ensures that the evaluated failures can be associated with the introduced Skill changes rather than with preexisting task difficulty or unrelated agent behavior.

C.4 Evaluation Protocol

For each retained failure, we compare the modified Skill file with its original successful version and use the resulting diff as ground truth for root cause localization. Given a failed execution, an RCA method predicts the responsible Skill edits or chunks, which we compare against the ground-truth hunks using span overlap to compute precision, recall, and F1.

We evaluate each Skill family using k-fold cross validation, where $k = \min(5, \mathcal{N}_{\text{fail}})$ and $\mathcal{N}_{\text{fail}}$ denotes the number of failed executions in that family. For each fold, the held-out failure is used for RCA judge, while the remaining executions are used to compute the suspiciousness prior. Precision measures the fraction of predicted edits that correspond to injected changes, while recall measures the fraction of injected changes that are successfully recovered. The results in Table 1 report the mean performance across folds.

C.5 Edit Analysis

Table 2: Edit hunk statistics by variant

Variant	Diffs	Total edit hunks	Mean hunks/diff	Traces analyzed
Wrong Content	16	33	2.06	238
Contradiction	10	11	1.10	160
Ambiguity	8	14	1.75	132

Table 2 summarizes the number of Skill-file diffs, edit hunks, and execution traces analyzed for each error category. Wrong Content contains 16 diffs across 238 traces, with 33 edit hunks in total and an average of 2.06 hunks per diff. Contradiction contains 10 diffs across 160 traces and averages 1.10

hunks per diff, indicating that most variants involve a single localized edit. Ambiguity lies between the two, with 8 diffs across 132 traces and an average of 1.75 hunks per diff.

D SKILL KNOWLEDGE GRAPH CONSTRUCTION

We construct a structured Knowledge Graph (KG) for each Skill file to represent individual instructions and the relationships among information distributed across different parts of the file. Across all experiments, the KG is constructed using GPT-5.5-mini. The graph contains three components: *entities*, *statements*, and *edges*. Entities represent concrete objects or concepts referenced in the Skill, such as parameters, resources, actions, conditions, or domain specific terms. Statements capture typed instructions or information, while edges encode relationships among these statements.

D.1 STATEMENT EXTRACTION AND TYPING.

We use a structured LLM call to decompose the Skill file into atomic statements, assign each statement a semantic type, extract its associated attributes(slots), and identify the entities it references. Each instruction, fact, or rule is represented as an individual statement, rather than treating an entire paragraph or section as a single node.

Different statement and directive type uses for our experiment is described in Table 3.

Table 3: Statement types used to represent heterogeneous information in a Skill file.

Statement Type	**Slots**	**Description / Example**
DeonticConditional	trigger, action	Conditional obligation or prohibition, e.g., "If X, perform Y."
GradedConstraint	action, operator, threshold, unit	Measurable constraint, e.g., "Use at most 3 retries."
Preference	preferred, dispreferred, exception	Soft preference between alternatives, e.g., "Prefer X over Y unless Z."
Procedure	ordered_steps, gates	An ordered execution step and its preconditions.
MannerModifier	parent_stmt, quality	Specifies how another action should be performed.
DeclarativeFact	subject, claim	A checkable fact, value, boundary condition, or described behavior.
Definition	term, definiens, scope	Defines a term used elsewhere in the Skill.
Capability	resource, availability	Specifies whether a tool, API, resource, or feature is available.
Example	input, expected_output, illustrates_ref	An example or walkthrough illustrating another directive.
MetaPolicy	governs_ref, resolution	Specifies how conflicts or priorities between directives are resolved.
Scope	condition, applies_to_ref	Defines when or where another directive applies.
Goal	intent	Describes the intended goal or purpose of a section.
RawSpan	–	Fallback for content that cannot be reliably mapped to another statement type.
NoOp	–	Narrative, heading, or navigation text without checkable instructional content.

This type specific representation preserves the structure relevant to different forms of Skill instructions. For example, the directive 'use at most 3 retries" is represented as a `GradedConstraint` with `action` = 'retry", `operator` = $\leq$, and `threshold` = 3. In contrast, 'if the request fails, retry the operation" is represented as a `DeonticConditional`, with `trigger` = 'the request fails" and `action` = "retry the operation". This structured representation allows different instruction types to retain the attributes needed for downstream analysis.

D.2 Entity Extraction

Alongside statements, we extract lightweight entity nodes that capture concrete elements likely to appear in execution traces. We consider six entity types:

$$\{\texttt{ApiCall}, \texttt{Argument}, \texttt{Resource}, \texttt{Condition}, \texttt{Action}, \texttt{Term}\}.$$

Each type captures a different form of execution-relevant information. `ApiCall` represents literal function, tool, or command names that may be invoked by the agent, such as `github_create_issue` or an aggregation call such as `resample('D').max()`. `Argument` captures parameters or flags associated with such calls, for example `response_format="json"` or a pagination `limit`. `Resource` denotes named files, tables, or other data artifacts, such as `tickets.json` or `answers.json`. `Condition` represents triggers or comparison boundaries that determine when an instruction applies, such as 'request fails" or 'body character count $\geq$ 700". `Action` captures the concrete behavior prescribed by a statement, such as `retry`, `archive`, `round up`, or `escalate`. Finally, `Term` represents domain-specific labels whose meaning is defined within the Skill, such as retention codes, compensation zones, or format categories. Numeric values and distinctive quantities are also retained when they provide useful anchors for matching Skill content to execution traces.

Each statement is linked to the entities it references through a `usesEntity` relation. These links are constructed deterministically by matching entity surface forms and aliases against the statement text and its extracted attributes. During trace analysis, the presence of these entities in agent actions, errors, or execution summaries provides an additional signal for determining whether the corresponding statement was active in a given execution.

D.3 Source Grounding.

Since statement segmentation and classification are LLM generated, we explicitly ground every extracted statement in the original Skill file. Each statement stores its exact source text together with the corresponding file location:

$$\{\texttt{raw_text}, \texttt{source_line}, \texttt{source_line_end}, \texttt{chunk_id}\}$$

For each extracted statement, the LLM is required to return an exact quotation from the Skill file. We then independently match this span against the original text to recover its source location, ensuring that downstream analysis relies on the underlying Skill rather than an LLM generated line reference.

We also track coverage over the complete Skill file. Non instructional content is represented as `NoOp`, while spans that cannot be assigned a semantic type with sufficient confidence are retained as `RawSpan`. This ensures that portions of the Skill are not silently omitted during graph construction.

D.4 Relation Construction.

After extracting statement and entity nodes, we construct edges between statements to capture relationships among instructions appearing at different locations in the Skill. We divide edge construction into *deterministic* and *LLM-derived* relations.

Deterministic relations are added when the connection can be established without semantic interpretation. These include explicit references(**reference**), procedural ordering (**precedes**), structural links induced by headings or shared entities, and contradictions between compatible numerical constraints. Numerical constraints are compared using their structured `operator`, `threshold`, and `unit` fields rather than surface-level textual similarity.

Relations requiring semantic interpretation are identified using a separate structured LLM call over the original Skill text and the extracted statements. We consider the following relation types:

- **depends on**: one statement relies on a condition or action established by another;
- **refines**: one statement further specifies how another instruction should be interpreted or executed;

- **scopes**: one statement defines the applicability of another;
- **illustrates**: an example demonstrates another instruction;
- **overrides**: one statement takes precedence over another; and
- **contradicts**: two statements provide incompatible guidance for the same situation.

Because this analysis is performed at the document level, the graph can recover relationships even when related instructions use different terminology.

D.4.1 EDGE REPRESENTATION.

Each relation is represented as an edge

$$e = (s_i, r, s_j),$$

where s_i and s_j are statement nodes and r denotes the relation between them. In addition to its relation type, an edge records whether it was obtained deterministically or through an LLM:

```
relation
source_uid
target_uid
derivation: "deterministic" | "llm"
properties
```

For LLM-derived relations, the edge additionally retains evidence and a confidence score. This provenance makes it possible to distinguish mechanically established relationships from those inferred through semantic reasoning.

E AGENT ATTRIBUTION AND LOGGING

Execution traces typically record what the agent did and observed during a task, but not what information drove each action. This missing provenance matters for failure analysis, since an incorrect action may arise from a Skill directive, the task instruction, an earlier observation, or the agent's own reasoning. We therefore enrich each trace with structured *action attribution* by rerunning the same task query and requiring the agent to record the source(s) influencing each action.

E.1 ATTRIBUTION TAXONOMY.

For an action a_t performed at step t, we associate one or more justifications describing the information that influenced the action. Each justification belongs to one of four categories:

- **Skill**: the action is influenced by a rule, directive, example, or other information provided by the Skill file.
- **Task Instruction**: the action follows information explicitly specified in the task instruction provided by the user, without additional influence from the Skill.
- **Prior Observation**: the action is driven by information obtained from an earlier observation in the current execution trajectory.
- **Own Judgment**: the action results from the agent's own reasoning when none of the above sources directly explain the decision.

The corresponding attribution type is explicitly represented in our implementation as:

```
basis: Literal[
"Skill",
"task_instruction",
"prior_observation",
"own_judgment"
]
```

E.2 CAUSAL ATTRIBUTION.

We aim to capture sources that *influence* an action rather than sources that are merely relevant to it. For this purpose, each attribution can additionally contain a `counterfactual` describing how the specific action would have changed had the attributed source not been available.

For Skill, Task Instruction, and Prior Observation attributions, we additionally record a *"source-quote"*, corresponding to the source information used by the agent. The implementation requires this evidence to correspond to the original source rather than a retrospective paraphrase. Together, an action justification is represented as

$$J_t = (\texttt{basis}, \texttt{source_quote}, \texttt{counterfactual}).$$

The relevant schema is summarized below:

```
class _Justification(BaseModel):
    basis: Literal[
        "Skill",
        "task_instruction",
        "prior_observation",
        "own_judgment"
    ] = Field(
        description=(
            "Where this action's rationale actually "
            "came from. Check independently, in this "
            "order, rather than defaulting to whichever "
            "is easiest to justify: (1) does a loaded "
            "Skill's own rule/example/table/directives "
            "specifically shape this action, not just "
            "topically relevant, but the reason a "
            "DIFFERENT action wasn't taken, even if "
            "the task instructions alone would also "
            "have led here on their own? If so, "
            "basis='Skill'. (2) 'task_instruction' "
            "when the Skill contributes nothing beyond "
            "what the instructions already state "
            "outright. (3) 'prior_observation' when "
            "an earlier tool result in this run is "
            "what drove it. (4) 'own_judgment' only "
            "when none of the above applies."
        )
    )

    source_quote: Optional[
        Union[str, list[str]]
    ] = Field(
        default=None,
        description=(
            "Exact, character-for-character substring "
            "of the named source (the Skill text, the "
            "task description, or an earlier tool "
            "observation in THIS run) -- located by "
            "exact string match, not read from memory. "
            "Required whenever basis != 'own_judgment'; "
            "null when basis == 'own_judgment'. If more "
            "than one separate, non-adjacent line from "
            "the SAME source jointly justify this one "
            "action under this one counterfactual "
            "(e.g. a rule stated in one section plus "
            "a value/table row from a different "
            "section), pass a list of exact substrings "
            "instead of one -- do NOT merge them into "
            "a single quote string, since a merged "
            "quote is no longer a verbatim substring "
```

```
            "of anything and will fail to resolve. "
            "Prefer a single string when one quote "
            "already suffices."
        ),
    )

    counterfactual: Optional[str] = Field(
        default=None,
        description=(
            "The causal test for whether this citation "
            "is real rather than decorative. State "
            "concretely what THIS SPECIFIC action would "
            "have been instead, had this source not "
            "applied -- a different tool, different "
            "parameter, different value, different "
            "file, a skipped step, etc. If your honest "
            "answer is 'the action would have been "
            "exactly the same either way,' this "
            "justification fails the test, remove it "
            "from the list rather than including it. "
            "If the source is plausibly relevant but "
            "you cannot honestly construct a concrete "
            "counterfactual for it, leave this null "
            "rather than omitting the citation "
            "entirely, a source recorded with an "
            "uncertain causal link is more useful "
            "downstream than one silently dropped; "
            "it will be kept for visibility but "
            "excluded from automated scoring."
        )
    )
```

E.3 Action Context

In addition to attribution, we collect auxiliary information that can facilitate downstream failure analysis, however they are not used in current experiment but can help agent developer for better debugging. Specifically, each action context contains (i) a set of *justifications* describing the sources influencing the action, (ii) an *expected result* describing the agent's falsifiable expectation of what the action should produce, and (iii) *values* containing concrete resolved quantities or parameters used by the action. The resulting schema is:

```
class _ActionContext(BaseModel):
justifications: list[_Justification]
expected_result: str
values: dict[str, str]
```

For example, if an action applies a threshold obtained from a Skill directive, the attribution can identify the corresponding Skill passage, while `values` can record the concrete threshold used by the action. This separates the *source* of the decision from the concrete values used during its execution.

Schema-Enforced Attribution in OpenHands. We implement this mechanism by modifying the OpenHands agent harness. Rather than reconstructing attribution after execution or relying solely on free-form reasoning traces, we augment the schema exposed to the model for tool calls. Specifically, the harness wraps OpenHands' tool-schema construction and injects an `action context` field whose type is the structured schema described above.

A simplified excerpt of the harness modification of openhand sdk as per the commit on july 2026 is shown below:

```
import openhands.sdk.tool as _oh_tool
def _install_schema_enforced_action_context():
...
def _wrapped(action_type):
base = _original(action_type)
...

```
```
    augmented = type(
        target_name,
        (base,),
        {
            "action_context": Field(
                description=
                    _ACTION_CONTEXT_FIELD_DESCRIPTION
            ),
            "__annotations__": {
                "action_context": _ActionContext
            },
        },
    )

    return augmented

_oh_tool._create_action_type_with_summary = _wrapped
```

The field presented to the model is explicitly described by the harness as:

```
_ACTION_CONTEXT_FIELD_DESCRIPTION = (
"REQUIRED on every call. Structured debugging "
"context for this action, see each sub-field's "
"own description for details."
)
```

Consequently, attribution is requested as part of the tool-call generation process itself. This allows the agent to provide the debugging context at the point at which an action is selected, rather than requiring a separate model to retrospectively infer why an action was taken.

F Analysis of Attribution Performance

F.1 Performance of action grounding

To validate the reliability of our runtime attribution scheme, we sampled 360 attributed agent actions, comprising 100 examples each for Skill, task instruction, and prior observation, and 60 examples for the agent's own judgment. To avoid attribution leakage, we removed the attribution based metadata from the execution traces before evaluation, ensuring that the attributed source and its supporting quote were not exposed to the evaluator through the action trace itself. We then used GPT-5.6 Sol with high reasoning effort to assess whether each attributed source causally supported the corresponding action. For own-judgment cases, we instead evaluated whether the action reflected a reasoning, planning, verification, or decision step originating from the agent itself. We use a strict criterion in which an attribution is considered correct only when the cited source directly explains a material part of the action; cases with only indirect or partial support are not counted as correct. As shown in Table 4, the attribution accuracy is 91% for Skill, 91% for task instructions, 97% for prior observations, and 78% for own judgment, yielding an overall accuracy of 89%. These results indicate that the recorded attribution generally provides a reliable representation of the context underlying agent actions, while distinguishing the agent's own judgment from externally grounded actions remains comparatively more challenging.

Table 4: Validation accuracy of runtime action attribution under a strict, leakage-free evaluation.

Attribution Source	Accuracy (%)
Skill	91
Task Instruction	91
Prior Observation	97
Own Judgment	78
Overall	**89**

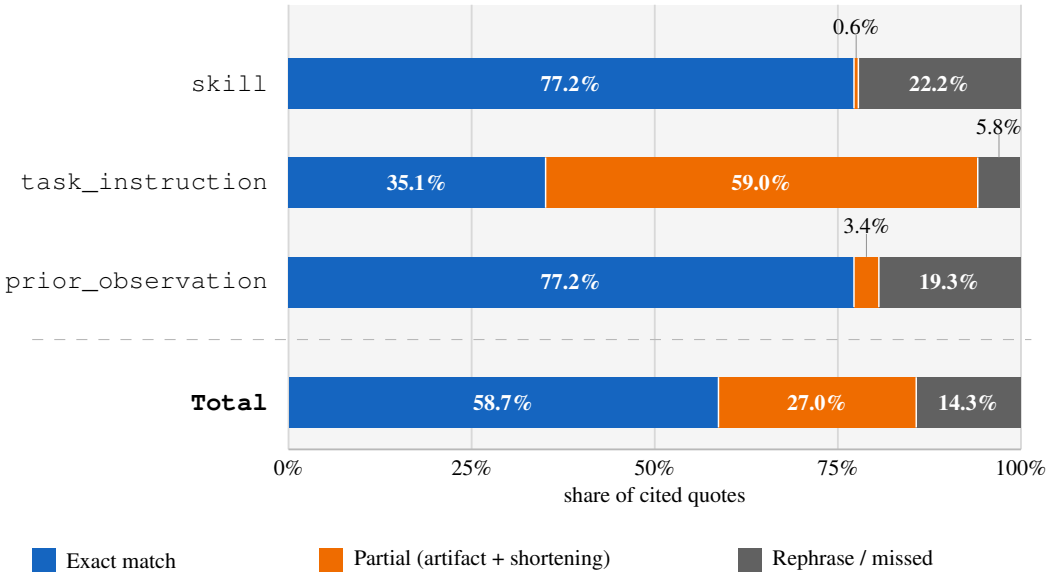

Figure 7: Traceability of quotes cited by the runtime agent, by attributed source. Bars give the share of quotes recovered verbatim, recovered only after shortening or newline artifacts are normalised, or not traceable to the cited source at all.

F.2 Performance of the context attribution

In our attribution scheme, alongside identifying the source supporting each action, the agent also records the corresponding context quote from the source. Since our attribution analysis relies on these quotes, we evaluate how reliably they can be grounded in the attributed source. For each quote, we use regular-expression matching against three candidate sources: the task instruction, the Skill file, and the text observed earlier in the execution. We classify the result as an *exact match* when the quoted text is recovered verbatim, a *partial match* when it remains recoverable after accounting for shortening or inserted line breaks, and *unmatched* when the quote is paraphrased or cannot be located.

Figure 7 summarizes the results. For Skill attributions, 77.2% of quotes match exactly and only 0.6% are partial, leaving 22.2% unmatched. Task-instruction quotes show a different pattern: 35.1% match exactly, while another 59.0% are recoverable after normalization, leaving only 5.8% unmatched. Prior-observation quotes are closer to Skill attributions, with 77.2% exact matches and 19.3% unmatched. Overall, 58.7% of all cited quotes are exact matches and a further 27.0% are recoverable after normalization, meaning that 85.7% of attribution quotes can be traced back to their reported source.

G AGGREGATE ANALYSIS AND SUSPICIOUSNESS ESTIMATION

Our aggregate analysis maps historical execution logs to statements in the Skill Knowledge Graph (KG) and estimates how strongly each statement is associated with failures. The analysis consists of three steps: (1) determining which Skill statements are active in each execution, (2) computing statement-level suspiciousness from the resulting incidence matrix, and (3) expanding individual suspicious statements into sets of related directives using the KG.

G.1 TRACE REPRESENTATION

For each execution log l_i, we extract the execution outcome together with information describing the task and agent behavior. In particular, a trace is represented as

$$l_i = (q_i, a_i, r_i, e_i, y_i, c_i),$$

where q_i is the user query, a_i is the agent's final action summary, r_i is the final response, e_i contains execution errors, $y_i \in \text{pass}, \text{fail}$ is the execution outcome, and c_i denotes the structured action context collected during execution.

We use two views of the log for statement activation. The *behavior context* contains the action summary, errors, and final response, and is used for deterministic entity matching. The broader *execution context* additionally contains the user query and is used when reasoning about whether a directive's triggering condition was satisfied.

G.2 STATEMENT ACTIVATION

For each Skill directive d_j and execution log l_i, we compute a binary activation indicator

$$\mathcal{I}_{ij} = \mathcal{I}(d_j, l_i) \in \{0, 1\},$$

where $I_{ij} = 1$ indicates that statement d_j was active during execution l_i. Activation is determined using complementary evidence from entity matching, agent attribution, and trigger evaluation.

Entity-based Activation. Each statement in the KG is associated with a set of entities extracted from its text and structured slots. We search for these entities in the execution log to identify candidate statements that may have influenced the execution. API calls, arguments, and resources appearing directly in the agent's behavior context provide a stronger action-specific matching signal.

Formally, let $\mathcal{E}(d_j)$ denote the entities associated with directive d_j. Entity evidence provides an initial candidate function

$$C(d_j, l_i) \in \{0, 1\},$$

indicating whether sufficient entities associated with s_j occur in the agent's observed behavior.

Attribution-based Activation. When action attribution is available, we additionally use the agent's own attribution as direct evidence of Skill usage. For each Skill attribution, the quoted source span is deterministically located in the original Skill file and mapped to the KG statement(s) whose source ranges overlap the cited span.

If an attribution from execution l_i resolves to directive d_j, we mark

$$\mathcal{I}(d_j, l_i) = 1.$$

We only use attributions corresponding to executed actions and for which a valid counterfactual was provided. This prevents rejected tool calls or weak, non-causal citations from directly influencing suspiciousness estimation.

Trigger-based Activation. For conditional statements such as `DeonticConditional` and `Scope`, the occurrence of an entity alone does not justify activating the associated directive; its applicability depends on whether the explicit trigger or condition is satisfied.

For candidate statements containing a `trigger` or `condition` slot, we therefore perform an additional NLI-based check:

$$T(d_j, l_i) = \text{NLI}\left(\text{condition}(d_j), \text{context}(l_i)\right).$$

The NLI model determines whether the execution provides sufficient evidence that the corresponding condition was satisfied. Candidate statements without an explicit trigger are activated directly, whereas triggered statements are activated only when the trigger check succeeds.

The resulting activation can therefore be viewed conceptually as

$$\mathcal{I}(d_j, l_i) = \begin{cases} 1, & \text{if } d_j \text{ is directly supported by valid attribution,} \\ C(d_j, l_i) \cdot T(d_j, l_i), & \text{if } d_j \text{ contains a trigger,} \\ C(d_j, l_i), & \text{otherwise.} \end{cases}$$

This process produces the statement–execution incidence matrix

$$\mathcal{I} \in \{0, 1\}^{|\mathcal{S}| \times |\mathcal{L}|},$$

where rows correspond to Skill statements and columns correspond to historical executions.

G.3 Ochiai Suspiciousness

Given the incidence matrix, we estimate the suspiciousness of each statement using the Ochiai coefficient

For each directive d_j, we compute

$$e_{\text{fail}}(d_j) = \sum_{l_i : y(l_i) = 0} \mathcal{I}(d_j, l_i)$$

and

$$e_{\text{pass}}(d_j) = \sum_{l_i : y(l_i) = 1} \mathcal{I}(d_j, l_i).$$

Let

$$\mathcal{N}_{\text{fail}} = |\{l_i : y_i(l_i) = 0\}|$$

denote the total number of failing executions. The Ochiai suspiciousness of d_j is

$$\pi(d_j) = \frac{e_{\text{fail}}(d_j)}{\sqrt{\mathcal{N}_{\text{fail}} \cdot \left(e_{\text{fail}}(d_j) + e_{\text{pass}}(d_j)\right)}}.$$

The score lies in $[0, 1]$. A high score indicates that a directive is frequently active in failing executions while being relatively uncommon in successful executions. In particular, if a directive is active in every failing execution and no successful execution, its Ochiai score is 1.

For example, consider $\mathcal{N}_{\text{fail}} = 4$. Suppose statement d_1 is active in all four failing executions and no successful execution:

$$e_{\text{fail}}(d_1) = 4, \qquad e_{\text{pass}}(d_1) = 0.$$

Then,

$$\pi(d_1) = \frac{4}{\sqrt{4(4+0)}} = 1.$$

In contrast, suppose d_2 is also active in all four failures but additionally appears in four successful executions:

$$\pi(d_2) = \frac{4}{\sqrt{4(4+4)}} \approx 0.707.$$

Thus, although both statements cover the same failing executions, d_1 provides stronger evidence because its activation is more specific to failures.

The resulting scores define the directive-level prior which captures historical evidence about which Skill statements are associated with failures.

G.4 KG-Guided Beam Search for Directive Sets

While the instruction based Ochiai considers each directive independently. However, some failures may arise only when multiple directives are simultaneously applicable. Two statements may individually occur across both successful and failing executions, yet their *joint activation* may occur predominantly in failures. Exhaustively evaluating every subset of $|\mathcal{S}|$ statements can require evaluation on combinatorial combinations. We therefore use a KG-guided beam search to identify suspicious combinations efficiently.

G.4.1 Set-level Activation.

For a candidate set of statements $\mathcal{C} \subseteq \mathcal{S}$, we define joint activation using intersection:

$$\mathcal{I}(\mathcal{C}, l_i) = \prod_{d_j \in \mathcal{C}} \mathcal{I}(j_j, l_i).$$

Hence, $\mathcal{C}$ is active in execution l_i only when *every* statement in the set is active in that execution.

We then compute

$$e_{\text{fail}}(\mathcal{C}) = \sum_{l_i : y(l_i)=0} \mathcal{I}(\mathcal{C}, l_i)$$

and

$$e_{\text{pass}}(\mathcal{C}) = \sum_{l_i : y(l_i)=1} \mathcal{I}(\mathcal{C}, l_i),$$

and define its set-level suspiciousness as

$$\pi_c(\mathcal{C}) = \frac{e_{\text{fail}}(\mathcal{C})}{\sqrt{N_{\text{fail}}\left(e_{\text{fail}}(\mathcal{C}) + e_{\text{pass}}(\mathcal{C})\right)}}.$$

The joint score therefore captures failure patterns that cannot be identified reliably from either directive independently.

KG-Constrained Expansion. Rather than considering arbitrary statements as additions to a candidate set, we constrain expansion using the Skill KG. Let

$$\mathcal{N}(\mathcal{C}) = \bigcup_{d \in \mathcal{C}} \mathcal{N}(d) \setminus \mathcal{C}$$

denote the set of statements connected by a KG edge to at least one current member. At each expansion step, only statements in $\mathcal{N}(\mathcal{C})$ are considered.

Thus, every candidate group explored by the search contains statements connected through an explicit relation in the Skill KG, such as `depends on`, `contradicts`, `references`, `refines`, or another available KG relation. This prevents the search from constructing arbitrary combinations solely because two statements happen to co-occur in historical failures.

Beam Search. The search begins with a single seed d^*. For every candidate set $\mathcal{C}$ currently in the beam, we enumerate each possible one-hop undirected KG expansion

$$\mathcal{C}' = \mathcal{C} \cup \{d'\}, \qquad d' \in \mathcal{N}(\mathcal{C}),$$

and compute $\pi_c(\mathcal{C}')$. After evaluating all expansions at the current depth, candidate sets are ranked according to their set-level Ochiai score and only the top B sets are retained:

$$\mathcal{B}_{t+1} = \mathrm{TopB}\left(\{\mathcal{C} \cup \{d'\} : \mathcal{C} \in \mathcal{B}_t, d' \in \mathcal{N}(\mathcal{C})\}\right),$$

where B denotes the beam width.

Seed Selection and Failure Coverage. A single search may identify a highly suspicious group that explains only a subset of the observed failures. We therefore repeat the expansion procedure using different seeds.

For the first search, seed selection favors statements with high failure coverage, using individual Ochiai as a secondary signal. After a group has been identified, we maintain the set of failing executions already covered by previously selected groups. Subsequent seeds prioritize statements that cover the largest number of *previously unexplained* failures, followed by their overall failure coverage and individual Ochiai score.

Conceptually, if $\mathcal{F}_{\mathrm{covered}}$ denotes failures explained by previous groups, the next seed prioritizes

$$\left|\mathcal{F}(d) \setminus \mathcal{F}_{\mathrm{covered}}\right|,$$

where $\mathcal{F}(d)$ denotes the failing executions in which statement s is active.

This iterative procedure produces a collection of suspicious directive sets that jointly explain different failure modes while avoiding exhaustive enumeration of all possible subsets of the Skill.

H Ablation of Co-occurrence Merging for Root Cause Analysis

Figure 8 presents an ablation of co-occurrence merging for RCA. Its effect is model dependent. For Opus-4.6, merging improves Wrong Content (0.842 vs. 0.789 F_1) and Contradiction (0.848 vs. 0.841), with the largest gain on Wrong Content. GPT-5.5 shows a different pattern, with improvement only on Ambiguity (0.548 vs. 0.535 F_1). This suggests that RCA judges use the suspiciousness prior differently: Opus-4.6 appears to benefit more from co-occurrence-based scores, whereas GPT-5.5 may rely more on its own reasoning over individual directives. Nevertheless, co-occurrence signals can provide developers with additional evidence about groups of directives that may jointly contribute to failures.

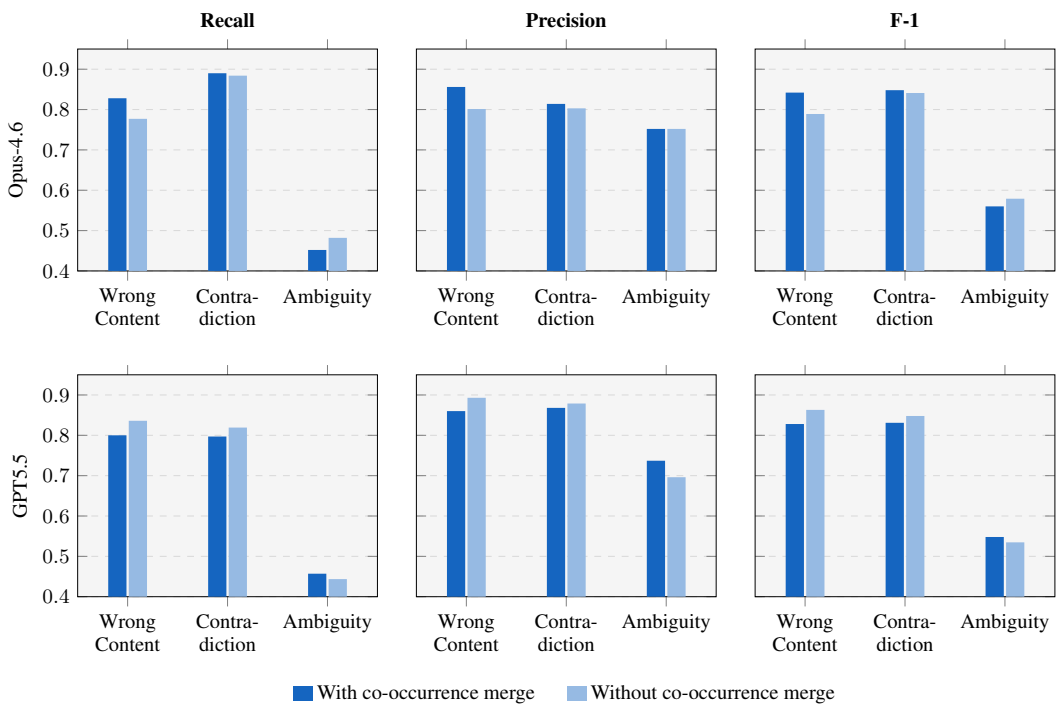

Figure 8: Effect of co-occurrence merging on recall, precision, and F-1 across error categories for Opus-4.6 (top) and GPT5.5 (bottom).

I LLM Judge Prompt and Failure Trace Analysis

The final RCA stage uses an LLM judge to select the Skill passages most likely responsible for a specific failure. The judge operates over the candidate shortlist generated by the preceding stages. We report below the prompts used in our implementation and specific evidence associated with trace.

I.1 Failure Trace Analysis

For execution traces associated with a failure, we extract four components as additional context for the RCA judge: the user query, all agent actions, the extracted error context, and the final message. Together, these constitute the trace-level evidence used for failure analysis. Across the log-based RCA baselines, the judge uses this evidence together with additional Skill content to identify potentially faulty directives. In our method, the same trace-level evidence is paired with a candidate list of directives, each accompanied by its suspiciousness prior score, to guide the final localization.

I.1.1 System Prompt

The following prompt is defined by `SYSTEM_PREAMBLE`, defining the persona of the RCA judge.

```
You are a root cause analyst. An agent followed a Skill guide (SKILL.md
    ) and produced a wrong answer on a task. One or more passages in
    the Skill file contain defects -- incorrect facts, contradictions,
    or vague rules -- that caused the failure.

You will be given a shortlist of candidate passages from the Skill file
     and one failing run transcript. Identify which candidate passage(s
    ) are responsible for the failure. Base your answer only on what
    the transcript explicitly shows -- values the agent computed, rules
     it applied, the wrong answer it produced.
```

I.1.2 MULTI-SELECT INSTRUCTION

The following instruction is appended through _MULTI_SELECT_NOTE and highlight that Judge should identify all faulty directives.

```
If none of the candidates plausibly explain the failure, say so in the
    summary and report zero findings.

This is NOT a "pick exactly one" question. Treat it like a multi-select
     multiple-choice question: if the transcript is genuinely
    consistent with more than one candidate contributing to (or
    independently explaining) the failure -- e.g. two related rules
    that together produced the wrong behavior, or two candidates that
    are each independently sufficient and you cannot tell which one the
     agent actually relied on -- report ALL of them as separate
    findings. Do not artificially collapse to a single "the root cause"
     when the evidence supports several; under-reporting a real co-
    culprit is a worse mistake here than over-reporting, as long as
    each finding you keep is still individually grounded in the
    transcript per the rule above.

Specifically check for a COMPANION restatement once you've found one
    suspect rule: Issues in the Skill files frequently states the same
    defective rule TWICE in one Skill file -- once as the primary rule,
     and again as its own short restatement/warning elsewhere (often
    under a heading like "Common mistakes" or "pitfalls"), phrased as "
    don't do X" where X is the CORRECT behavior the primary rule now
    forbids. Before finalizing your findings, scan the rest of the
    shortlist for a second candidate that restates the SAME specific
    fact -- the same number, threshold, or direction -- as the one you
    already flagged. A true companion names that same specific value (
    even in different wording); it is not merely on the same general
    topic or in the same section. If a candidate only discusses the
    topic in general terms (e.g. a broader warning about precision/
    tolerance/consistency) without itself restating the specific number
     or direction you flagged, it is NOT a companion -- leave it out.
    Report a real companion as its own separate finding, not folded
    into the first; do not report a same-topic candidate that fails
    this test just to have a second finding.
```

I.1.3 OCHIAI-SCORE GUIDANCE

The following prompt, defined by _OCHIAI_NOTE, specifies how the judge uses the historical suspiciousness scores.

```
Each candidate carries an "ochiai=" suspiciousness score (0 to 1): a
    STATISTICAL signal computed across EVERY subtask in this task
    family -- higher means that passage was active in FAILING runs and
    absent/rare in PASSING ones. This is real evidence and the primary
    ranking signal: the true root cause is very likely among the
    highest-ochiai candidates, so always search those first, and do not
     walk away with zero findings just because the transcript reads as
    ambiguous. If the top candidates are tied (or nearly so) and the
    transcript doesn't clearly favor one over the others, report the
    tied set rather than guessing a single one -- a wrong guess is
    worse than an honest hedge.

To REPORT a candidate, weigh both signals together, not either alone:
    (1) ACTIVE -- the transcript concretely shows the agent applying
    that exact rule/value/instruction, and doing so is consistent with
    the wrong output, not merely that the passage exists in the Skill
    file; and (2) ochiai -- a high score corroborates it, while a low
    score (relative to other candidates) should make you hesitate even
```

```
    if the agent technically touched it. One trap: an agent that
    applies a rule consistently and reasons its way to a coherent-
    looking answer is not thereby proven innocent -- the task was still
     scored wrong, so if a candidate's stated value is what the agent
    followed, that value stays a live suspect even when the agent's own
     math checks out. Do not report a candidate on ochiai alone with no
     sign it was active in this transcript, and do not report a
    candidate as active while ignoring that some other, higher-ochiai
    candidate fits the same evidence just as well or better -- unless
    the two are companions or independently sufficient (both below), in
     which case report both.

Once you've confirmed one candidate is active, check the rest of the
    shortlist for two more things before you finalize: a companion that
     restates the SAME specific rule/value/threshold elsewhere (
    different wording, section, or ochiai score -- still worth
    reporting, since ochiai is scored per-statement and a real
    companion can score very differently); and, separately, any OTHER
    candidate that would, on its own, via a genuinely different rule,
    also explain the same wrong output -- two unrelated defects can
    coexist in one file. Report every one you find this way. Don't
    force a second finding onto a candidate that's merely on the same
    topic without meeting one of these two tests.
```

I.1.4 FAILURE-SPECIFIC PROMPT

For each failing execution, the implementation dynamically constructs the following prompt. The placeholders are populated with the corresponding failure and candidate information at runtime.

```
--- FAILURE: {subtask.name} ---
{failure_excerpt}

{_OCHIAI_NOTE}--- CANDIDATE SHORTLIST ({len(shown)} candidates, ranked
    by Ochiai) ---
{candidate_block}

Identify which candidate(s) caused this failure. Start from the highest
    -ochiai candidates and confirm against the transcript above.
```

Each entry in `candidate_block` is constructed using the following template:

```
[Candidate {i}] {loc} [{prior value}]
{quote}
```

Here, `failure_excerpt` contains the execution trace as described in section I.1 for the failure, while `candidate_block` contains the Skill passages surfaced by the preceding RCA stages together with their locations and Ochiai scores(prior value).