Context or Capability? Debugging Agentic Workflows

Paulina Toro Isaza Saurabh Jha Yu Deng
IBM Software Innovation Lab
Yorktown Heights, NY 10598
{ptoroisaza, Saurabh.Jha}@ibm.com, dengy@us.ibm.com

Abstract

When a multi-step LLM agent fails, the terminal accuracy score says *that* something went wrong but is silent on *where* and *why*. We address this diagnostic problem with a structural causal framework that localizes faults to specific nodes in the workflow DAG and distinguishes failures of insufficient context from failures of inadequate decision-making, mapping each diagnosis onto a well-defined intervention. We validate on IT-Bench, a Kubernetes incident diagnosis benchmark, across three models. For GPT-OSS-120B, the framework traces failures to a long-context bottleneck mid-workflow; addressing it lifts end-to-end task accuracy from 59% to 74% — +15pp absolute / 27% relative improvement.

1 Introduction

LLM agents are increasingly deployed as multi-step workflows whose execution forms a directed acyclic graph (DAG): a retriever feeds a reasoner, which feeds a planner, which invokes tools. These workflows are typically evaluated only with binary end-to-end labels that say *that* something went wrong but not *where* or *why*. A reasoner failure might be a reasoning bug, a missing retrieval that left it with nothing to work on, or a silent upstream error cascading downstream; each demands a categorically different intervention.

This conflation is the central obstacle to systematic debugging. A single failed trace cannot separate sampling variance from systematic failure, and even aggregated failure rates mix two different causes: receiving insufficient context versus making poor decisions on adequate context. The standard framing of "the model got it wrong" collapses this distinction. Hence, our goal is to develop a debugging framework that, given a multi-node LLM workflow evaluated over a benchmark, localizes the fault to a specific node and determines whether the failure is due to insufficient context (upstream) or inadequate decision-making (local), so that the minimal effective intervention can be identified.

Our key insight is that the workflow DAG is a causal structure: the output of node j *causes* part of the input to node i whenever there is an edge from j to i. This causal structure determines how failures propagate, what interventions are well-defined, and which nodes are responsible for downstream errors. We exploit this structure by formalizing the workflow as a structural causal model (SCM) [13], decomposing each node into its *context* (everything the model sees) and *decision* (what the model does with it), and defining binary indicators C_i for context sufficiency and D_i for instrumental correctness (i.e., whether the output carries what downstream nodes need). The law of total probability then yields a decomposition at every node:

$$P(D_i) \;=\; \underbrace{P(D_i \mid C_i{=}1)}_{\mu_i}\cdot P(C_i) \;+\; \underbrace{P(D_i \mid C_i{=}0)}_{\lambda_i}\cdot(1 - P(C_i)). \qquad (\star)$$

This separates each node's observed performance into *mechanism quality* μ_i (how well the node converts sufficient contexts into correct decisions), *context supply* $P(C_i)$ (how often upstream delivers what is needed), and a *noise floor* λ_i (the rate of correct decisions despite insufficient context — via parametric knowledge or lucky sampling). Because LLMs are stochastic, $\mu_i < 1$ in general (good context does not guarantee success) and $\lambda_i > 0$ (bad context does not guarantee failure).

AI Agents for Discovery in the Wild (AID-Wild), Workshop at ACM CAIS 2026.

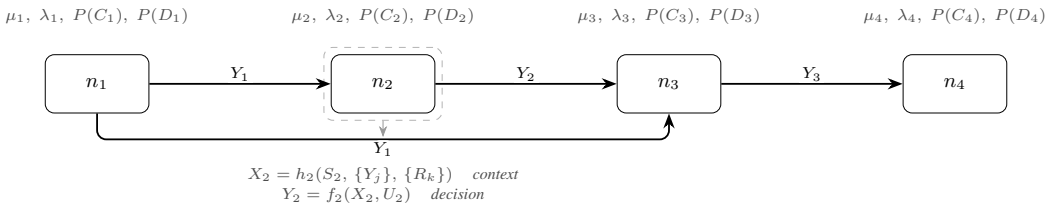

Figure 1: The context/decision framework. Each node n_i splits into context X_i and decision Y_i and is summarized by $(\mu_i, \lambda_i, P(C_i), P(D_i))$. Notation and the cascade are defined in §2.1–§2.3.

The framework's diagnostic power rests on the *separation property*: $\mu_i \gg \lambda_i$ for non-trivial tasks. When separation holds, the decomposition reduces to $P(D_i) \approx \mu_i \cdot P(C_i)$ with bounded error, and structure functions propagate context sufficiency through the DAG, producing a recursive system solved forward in topological order.

Three results follow from the decomposition. It induces an *exhaustive failure taxonomy* (§2.7): every underperforming node falls into exactly one of context-limited, mechanism-limited, or joint failure, and the case dictates whether to fix the input, the mechanism, or an upstream ancestor. It gives *interventional semantics* (§2.6): editing a node's context or mechanism is a well-defined intervention answering "what would performance be if we *guaranteed* correct context?" And it yields a *backward diagnostic trace* (§2.8) that localizes root causes, derives labeling criteria for intermediate nodes, and decomposes end-to-end performance into a product of local mechanism rates.

Validation. We validate on IT-Bench [5], a real-time Kubernetes incident diagnosis benchmark, across three models of varying capability. For GPT-OSS-120B, the decomposition traces the bottleneck to a long-context processing failure mid-workflow; a targeted context-reduction intervention lifts end-to-end task accuracy from 59% to 74% — a 15pp absolute improvement (27% relative), providing a proof-of-concept validation that the framework correctly localizes faults and guides interventions (§3).

2 Methodology

We formalize the workflow DAG as an SCM in which nodes produce outputs that cause downstream inputs, define what "correct" means at intermediate nodes, derive the node decomposition, and develop its implications for debugging. Figure 1 illustrates the core idea.

2.1 The Workflow as a Structural Causal Model

An LLM workflow consists of nodes that consume upstream outputs and produce downstream inputs; orchestration platforms make this DAG structure explicit, and autonomous agents can be cast into the same formalism by abstracting execution traces into recurring phases.

Definition 1 (Workflow SCM)**.** *An LLM workflow is a structural causal model $\mathcal{M} = (\mathcal{G}, \mathbf{F}, \mathbf{U}, \mathbb{B})$:*

- *$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a DAG with LLM nodes $\mathcal{F} \subseteq \mathcal{V}$ and tool nodes $\mathcal{T} \subseteq \mathcal{V}$.*
- *$\mathbf{F} = \{f_i, h_i, \tau_k\}$ is a collection of structural equations (below).*
- *$\mathbf{U} = \{U_i\}_{i \in \mathcal{F}}$ are mutually independent exogenous noise variables ($U_i \sim \mathcal{U}_i$), representing each LLM's stochastic sampling.*
- *$\mathbb{B} = \{1, \ldots, K\}$ is a benchmark of task instances, each equipped with a terminal correctness criterion $J^{(k)} : \mathcal{Y}_T \to \{0, 1\}$.*

The criterion $J^{(k)}$ is a modeling choice that reflects available supervision: *gold labels*, $J^{(k)}(Y_T) = \mathbf{1}[Y_T = G^{(k)}]$; *human judgment*, $J^{(k)}(Y_T) = \mathbf{1}[\text{human rates } Y_T \text{ acceptable}]$; or *LLM-as-judge*, $J^{(k)}(Y_T) = \mathbf{1}[\text{judge model scores } Y_T \geq \theta]$. The entire framework (decomposition, taxonomy, backward trace) is parametric in J.

Each LLM node has two endogenous variables. The **context** X_i is everything the model sees (static prompt, parent outputs, accumulated history) assembled deterministically:

$$X_i \ := \ h_i\big(S_i,\ \{Y_j\}_{j \in \mathrm{Pa}(i) \cap \mathcal{F}},\ \{R_k\}_{k \in \mathrm{Pa}(i) \cap \mathcal{T}}\big)\,. \tag{1}$$

The **decision** Y_i is the model's stochastic output given its context:

$$Y_i \;:=\; f_i(X_i, U_i), \quad U_i \sim \mathcal{U}_i. \tag{2}$$

Tool nodes are approximately deterministic functions of their parent's decision: $R_k := \tau_k(Y_{\mathrm{pa}(k)})$. Because h_i and τ_k are deterministic, all stochasticity originates from $\mathbf{U}$. Each task instance k is run N times (independent draws from $\mathbf{U}$), yielding observations indexed (k, n).

2.2 Correctness at Intermediate Nodes

The terminal criterion J tells us whether the workflow produced an acceptable output. But debugging requires knowing whether *each intermediate node* did the right thing which is not self-evident at an intermediate node. A retrieval node's output is not "correct" or "incorrect" in isolation; it is correct insofar as it enables the downstream nodes to eventually produce the right final answer.

We define two binary indicators for each observation (k, n) at each node n_i.

Definition 2 (Decision correctness)**.** *The decision $Y_i^{(k,n)}$ is* instrumentally correct *($D_i^{(k,n)} = 1$) if it belongs to the set of outputs at node n_i that are compatible with eventual success on task k — i.e., if the information it carries is what downstream nodes need to produce a correct terminal output.*

Definition 3 (Context sufficiency)**.** *The context $X_i^{(k,n)}$ is* sufficient *($C_i^{(k,n)} = 1$) if it contains the information that a competent model would need to produce an instrumentally correct decision at node n_i for task k.*

These are *informational* properties: C_i asks whether the input has what is needed; D_i asks whether the output carries what is needed downstream. Crucially, because Y_i depends on both X_i and the noise U_i, the relationship between C_i and D_i is stochastic, not deterministic:

- $C_i = 1$ does not guarantee $D_i = 1$: the noise U_i may cause the model to produce an incorrect output despite having sufficient context.
- $C_i = 0$ does not guarantee $D_i = 0$: the noise U_i may lead the model to produce a correct output despite insufficient context (e.g., via parametric knowledge or lucky sampling).

The framework's diagnostic power rests on the *separation* between these two regimes, formalized in the next section.

Remark 1 (Gold ground truth)**.** *When a deterministic ground truth $G^{(k)}$ exists, $J^{(k)}(Y_T) = \mathbf{1}[Y_T = G^{(k)}]$ and decision correctness at each node asks whether the output preserves or produces the information needed to eventually reach $G^{(k)}$. This is the simplest setting; our experiments (§3) use it throughout.*

2.3 The Node Decomposition

Applying the law of total probability to the indicators C_i, D_i at any LLM node n_i gives the *node decomposition*:

$$P(D_i = 1) \;=\; \underbrace{P(D_i = 1 \mid C_i = 1)}_{\mu_i} \cdot P(C_i = 1) \;+\; \underbrace{P(D_i = 1 \mid C_i = 0)}_{\lambda_i} \cdot (1 - P(C_i = 1)). \tag{3}$$

The decomposition separates each node's observed performance into three quantities:

- **Mechanism quality** $\mu_i = P(D_i \mid C_i = 1)$: how well the node converts sufficient contexts into correct decisions. This is where the noise U_i acts: even with good context, stochastic sampling may produce failure, so $\mu_i < 1$ in general.
- **Context supply** $P(C_i)$: how often the node receives sufficient context from upstream.
- **Noise floor** $\lambda_i = P(D_i \mid C_i = 0)$: the rate at which the model produces correct decisions despite insufficient context whether via parametric knowledge, lucky sampling, or task simplicity.

Definition 4 (Separation property)**.** *A node n_i exhibits* context-sensitivity *if $\mu_i \gg \lambda_i$: the model's success rate is substantially higher with sufficient context than without.*

When separation holds, the decomposition reduces to the approximate product:

$$P(D_i) \;\approx\; \mu_i \cdot P(C_i), \quad \text{with error } \varepsilon_i := \lambda_i \cdot (1 - P(C_i)) \;\leq\; \lambda_i. \tag{4}$$

The error ε_i is *self-damping*: it is small whenever *either* λ_i is small (context-sensitive task) *or* $P(C_i)$ is high (context usually sufficient). It can only be large when both λ_i is non-negligible and context frequently fails — a condition that is itself diagnostic.

Separation is structurally expected for any well-designed workflow: if providing the right context does not substantially improve a node's output, the node has no business consuming that context. A violation of separation signals a labeling issue (C_i does not capture what actually matters), excessive reliance on parametric knowledge, or a task simple enough to solve without upstream information. We verify separation empirically in §3.

Remark 2 (Diagnostic value of λ_i)**.** *A non-zero λ_i is itself informative: it indicates that some correct decisions are produced despite insufficient context. It often reflects that the labeling approximation from the binary C_i label does not capture every pathway to a correct decision. This does not compromise the framework as long as ε_i remains small.*

2.4 How Context Propagates Through the DAG

Failures propagate through the graph via each node's dependency structure on its parents.

Definition 5 (Structure function)**.** *Each node n_i with parents $\mathrm{Pa}(i)$ has a monotone structure function $\phi_i : \{0,1\}^{|\mathrm{Pa}(i)|} \to \{0,1\}$ determining context sufficiency from parent decision correctness:*

$$C_i^{(k)} = \phi_i\Big(\{D_j^{(k)}\}_{j\in\mathrm{Pa}(i)}\Big)\,. \tag{5}$$

Common cases: AND-gate ($\bigwedge_j d_j$, every parent must succeed), OR-gate ($\bigvee_j d_j$, any parent suffices), or single-parent ($\phi_i(d)=d$).

Remark 3 (Granularity)**.** *A node may encapsulate a single LLM call or an entire sub-workflow. When a node encapsulates internal complexity, μ_i reflects the combined mechanism quality of everything inside it. In §3, Bootstrap encapsulates three LLM calls and their aggregation; the Final Report's structure function is an OR-gate over Bootstrap and Investigation.*

Proposition 1 (Context propagation)**.** *Given conditional independence of parent decisions:*

$$P(C_i = 1) = \sum_{\mathbf{d}\in\{0,1\}^{|\mathrm{Pa}(i)|}} \phi_i(\mathbf{d}) \prod_{j\in\mathrm{Pa}(i)} P(D_j)^{d_j}(1-P(D_j))^{1-d_j}. \tag{6}$$

Under AND: $P(C_i) = \prod_j P(D_j)$. Under OR: $P(C_i) = 1 - \prod_j (1 - P(D_j))$.

2.5 The Cascade

Combining Eq. 3 with Proposition 1 gives a recursion solved forward in topological order, with $P(C_i)$ at sources reflecting benchmark input adequacy. For a linear chain $n_1 \to \cdots \to n_T$ with $\phi_i(d)=d$, this telescopes under separation:

Theorem 1 (Chain factorization)**.** *When $\mu_i \gg \lambda_i$ at every node:*

$$P(D_T) \;\approx\; P(C_1)\cdot\prod_{i=1}^{T}\mu_i, \quad \textit{with error at most } \textstyle\sum_i \lambda_i. \tag{7}$$

The bound $\sum_i \lambda_i$ is conservative; the practical error is $\sum_i \varepsilon_i \ll \sum_i \lambda_i$ because each node's noise is damped by its own context supply. A single weak node $\mu_i = p \ll 1$ caps the entire pipeline at $P(D_T) \lesssim p$; OR-gated DAGs add redundancy.

2.6 Interventions

Two classes of intervention are well-defined on the SCM: replacing a node's context, or replacing the mechanism that produces its decision. Either lets us answer the counterfactual "what would this node's performance be if we *guaranteed* correct context?"

Definition 6 (Class I: context replacement)**.** *Replace the context assembly h_i with a constant x_i^* or a transformation $t(X_i)$, severing upstream dependencies. The post-intervention rate $P(D_i^*)$ measures mechanism quality independent of upstream performance. Context summarization, reduction, and augmentation are all Class I.*

Definition 7 (Class II: mechanism replacement)**.** *Replacing f_i with f_i^* (prompt change, model substitution, task decomposition, fine-tuning). The context assembly is unmodified.*

2.7 Exhaustive Failure Taxonomy

Given a performance threshold θ, an underperforming node ($P(D_i) < \theta$) falls into exactly one of three cases (Table 1), determined by which of μ_i and $P(C_i)$ are below θ. Case B subsumes two sub-causes that hypothesis generation (§2.9) disambiguates: if μ_i varies with a measurable context property (e.g., input length), the input is not processable in its current form; if μ_i is uniformly low, the mechanism lacks capability. In Case C, resolve context first — the mechanism estimate is conditioned on a small, non-representative subset and is unreliable.

Table 1: Failure taxonomy. Cases partition the failure space.

Case	Condition	Diagnosis	Intervention
A: Context-limited	$\mu_i \geq \theta$, $P(C_i) < \theta/\mu_i$	Mechanism OK; fault upstream	Class I or trace ancestors
B: Mechanism-limited	$\mu_i < \theta$, $P(C_i) \geq \theta$	Local: capability/processability	Class II or Class I transform
C: Joint	$\mu_i < \theta$, $P(C_i) < \theta$	Context corrupted; μ_i unreliable	Resolve context first

2.8 Backward Trace: From Terminal Failure to Root Cause

The decomposition induces a backward diagnostic procedure starting from the terminal node. If $P(D_T) < \theta$: (1) decompose to check if the failure is mechanism or context; (2) if $\hat{\mu}_T < \theta$, node T is the target (Case B); (3) otherwise the bottleneck is upstream, and each parent $n_j \in \mathrm{Pa}(T)$ is examined via ϕ_T and decomposed recursively. The trace terminates at a Case B node or at a source where $P(C_i) < \theta$.

2.9 From Diagnosis to Hypothesis

The backward trace identifies a target node and a failure case (A, B, or C). The next question is *why*: what property of the context or mechanism is responsible? The decomposition narrows the search space, but it does not pinpoint the cause. For example, a Case B failure means the problem is in how the node processes its input, not in the input itself. Three hypothesis generation methods bridge this gap, in increasing order of model dependence.

Metric differential analysis. Stratify the benchmark instances at node n_i by a measurable property of the context or decision (input token count, structural completeness, presence of a specific field) and compare μ_i across strata. A significant difference identifies the stratifying variable as a candidate cause. This method is cheap, interpretable, and operates on data already available or quickly calculated from the benchmark runs.

Cross-model differential analysis. When trajectories from multiple models are available, an LLM judge compares the contexts and decisions at n_i across models of differing capability. If a stronger model's context at node n_i systematically differs from the target model's in length, structure, or content, the difference suggests a concrete Class I intervention: use the stronger model's context pattern as the replacement context x_i^*.

Within-model differential analysis. An LLM judge is presented with paired examples of correct and incorrect decisions at n_i from the target model and asked to identify systematic differences. This method requires neither a reference model nor measurable signals, making it the most flexible of the three, but its output should be treated as a hypothesis to validate rather than a definitive explanation.

Metric differential analysis is the natural starting point. LLM-based methods are most valuable when the failure pattern is not captured by simple measurable properties. In both intervention case studies (§3), metric differential analysis identified the root cause and directly suggested the intervention.

2.10 Labeling

In practice, the binary indicators $C_i^{(k,n)}$ and $D_i^{(k,n)}$ are assigned by a labeling procedure that proceeds backward through the DAG:

1. **Terminal decision labels**: $\hat{D}_T^{(k,n)} = J^{(k)}(Y_T^{(k,n)})$ — directly from the correctness criterion.

2. **Intermediate decision labels**: $\hat{D}_i^{(k,n)}$ evaluated against the correctness criterion derived from the structure function ϕ_j of each child $j \in \mathrm{Ch}(i)$ — via rule-based matching, NLP metrics, or LLM-as-judge.
3. **Context labels**: $\hat{C}_i^{(k,n)} = \phi_i(\{\hat{D}_j^{(k,n)}\}_{j \in \mathrm{Pa}(i)})$ — computed mechanically from parent D labels via the structure function. For source nodes ($\mathrm{Pa}(i) = \emptyset$) or nodes receiving external inputs not captured by ϕ_i, $\hat{C}_i$ is assigned directly via rule-based matching or LLM-as-judge applied to X_i.

$\hat{D}_i = 1$ with $\hat{C}_i = 0$ is a valid observation representing a correct decision produced despite insufficient context (the noise U_i compensated for missing information). These observations contribute to the estimate of λ_i.

2.11 Putting It Together: The Diagnostic Pipeline

Algorithm 1 summarizes the end-to-end diagnostic procedure. Labeling and estimation are demand-driven: the backward trace derives labeling criteria for each intermediate node from the structure function ϕ_i of its children, so nodes are labeled only as the trace reaches them. A priority worklist (ordered by $\hat{P}(D_i)$ ascending) ensures the worst-performing nodes are investigated first, and a visited set prevents redundant work in diamond-shaped DAGs.

3 Experiments

3.1 Setup

We evaluate agent trajectories on IT-Bench [5], a real-time Kubernetes environment for diagnosing injected faults. The workflow under test is an agent that diagnoses incidents through three phases, forming the DAG in Figure 2 (Appendix A).

DAG structure. The workflow is decomposed into three nodes at the phase level (Remark 3). **Bootstrap** ($i = B$) encapsulates three LLM calls (entity selection, investigation summarization, change event analysis) which together select one to three candidate entities from observability and alert data. **Investigation** ($i = I$) encapsulates a sequential chain of entity judgment calls, each classifying an entity as primary failure, symptom, or exonerated. **Final Report** ($i = F$) aggregates bootstrap context and investigation judgments to produce the final diagnosis. The structure function ϕ_F is an OR-gate: the root cause entity can reach the final report via bootstrap or investigation.

Labeling. Node-level correctness labels are derived by propagating the ground truth root cause entity backward through the DAG. At each node, a decision is instrumentally correct if it correctly identifies or preserves the root cause entity. Labels are computed using rule-based matching and LLM-as-judge; the full labeling procedure is in Appendix B.

Models and data. The agent was originally evaluated on 17 incidents across 5 independent trials per model on three models of varying capability (Table 2); per-node counts are in Table 7 (Appendix C). To estimate diagnostic probabilities and capture variability from LLM generation, each independent trial was repeated 5 times. Context labels are derived from the first repetition of each trial ($\sim$85 observations per node); decision metrics pool all 5 repetitions ($\sim$425 observations per node). All reported intervals are $\pm 2\sigma$ Wilson CIs.

Table 2: Models evaluated

Model	Identifier	Params	IT-Bench (%)
o4-mini [10]	`o4-mini-2025-04-16`	NA	79.3 ± 1.6
GPT-OSS-120B [9]	`openai/gpt-oss-120b`	120B	58.6 ± 2.7
Granite 3.2 8B [4]	`ibm/granite-3-2-8b-instruct`	8B	15.1 ± 0.7

3.2 Diagnostic Probabilities

Table 3 reports diagnostic estimates for the node decomposition (Eq. 3) at each node. The three models exhibit qualitatively different failure profiles:

Algorithm 1 Structural Causal Debugging

Require: Workflow SCM $\mathcal{M}$; observations $\{(X_i^{(k,n)}, Y_i^{(k,n)})\}$ for all $n_i \in \mathcal{F}$, $k \in \mathbb{B}$, $n \in [N]$; correctness criterion J; threshold θ
Ensure: Set of root-cause nodes, each with failure case and intervention

$\mathcal{R} \leftarrow \emptyset$ ▷ root causes
$\mathcal{S} \leftarrow \emptyset$ ▷ visited nodes
INITIALIZE: for each terminal node n_T, label via $J^{(k)}(Y_T^{(k,n)})$, estimate $\hat{P}(D_T)$
$\mathcal{Q} \leftarrow$ priority worklist of all terminal n_T with $\hat{P}(D_T) < \theta$, ordered by $\hat{P}(D_i)$ ascending

while $\mathcal{Q} \neq \emptyset$ **do**
  $n_i \leftarrow \mathcal{Q}$.POP()
  if $n_i \in \mathcal{S}$ **then continue** ▷ skip if already visited
  end if
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{n_i\}$
  Label $\hat{C}_i^{(k,n)}$ via ϕ_i from parent $\hat{D}$ labels, or directly if n_i is a source node or receives external inputs; estimate $\hat{P}(C_i)$, $\hat{\mu}_i$, $\hat{\lambda}_i$

  if $\hat{\mu}_i \geq \theta$ **then** ▷ Case A: context-limited
    for each parent $n_j \in \mathrm{Pa}(i)$ **do**
      Derive labeling criterion for n_j from ϕ_i
      Label n_j: assign $\hat{C}_j^{(k,n)}, \hat{D}_j^{(k,n)}$
      Estimate $\hat{P}(D_j)$, $\hat{P}(C_j)$, $\hat{\mu}_j$, $\hat{\lambda}_j$
      if $\hat{P}(D_j) < \theta$ **then**
        $\mathcal{Q}$.PUSH(n_j)
      end if
    end for
  else if $\hat{P}(C_i) \geq \theta$ **then** ▷ Case B: mechanism failure
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(n_i, \text{Case B})\}$
  else ▷ Case C: joint failure
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(n_i, \text{Case C})\}$; enqueue parents as in Case A
  end if
end while

for each $(n_i, \text{case}) \in \mathcal{R}$ **do** ▷ address root causes by severity
  HYPOTHESIZE (§2.9): metric differential, cross-model differential, or LLM-as-judge
  INTERVENE (§2.6): design Class I or Class II intervention; re-run and re-evaluate
end for

Table 3: Context & decision probabilities at each node

	o4-mini			GPT-OSS-120B			Granite 3.2 8B		
	B	I	F	B	I	F	B	I	F
$P(C_i)$	$90.6^{+4.6}_{-8.3}$	$85.2^{+6.0}_{-9.1}$	$83.0^{+6.5}_{-9.4}$	$92.9^{+3.9}_{-7.8}$	$75.9^{+7.9}_{-10.2}$	$56.3^{+10.1}_{-10.7}$	$93.9^{+3.5}_{-7.6}$	$47.7^{+10.5}_{-10.3}$	$14.8^{+9.1}_{-6.0}$
$P(D_i \mid C_i{=}1)$	$94.7^{+1.9}_{-2.8}$	$94.7^{+1.9}_{-2.8}$	$98.1^{+1.0}_{-2.3}$	$81.8^{+3.6}_{-4.3}$	$80.0^{+4.0}_{-4.8}$	$96.4^{+1.7}_{-3.2}$	$84.2^{+3.4}_{-4.1}$	$29.0^{+6.6}_{-5.8}$	$77.0^{+8.9}_{-12.2}$
$P(D_i)$	$86.9^{+2.9}_{-3.6}$	$80.7^{+3.5}_{-4.0}$	$79.4^{+3.8}_{-4.4}$	$77.1^{+3.9}_{-4.4}$	$60.7^{+4.6}_{-4.8}$	$58.6^{+4.7}_{-4.8}$	$79.0^{+3.8}_{-4.3}$	$13.9^{+3.6}_{-3.0}$	$15.0^{+4.0}_{-3.3}$
ε_i	$1.2^{+1.6}_{-0.7}$	$0.0^{+0.9}_{-0.0}$	$0.0^{+1.0}_{-0.0}$	$1.2^{+1.6}_{-0.7}$	$0.0^{+0.9}_{-0.0}$	$2.1^{+1.9}_{-1.0}$	$0.0^{+1.0}_{-0.0}$	$0.0^{+0.9}_{-0.0}$	$2.6^{+2.2}_{-1.2}$

B=Bootstrap, I=Investigation, F=Final Report. All values in %; intervals are $\pm 2\sigma$ Wilson CIs.

o4-mini. Context correctness degrades from Bootstrap to Final Report while $P(D_i \mid C_i = 1)$ remains above 0.94 throughout, indicating a cascading context failure (Case A at each node). The mechanism is not the bottleneck; the pipeline compounds upstream context loss.

GPT-OSS-120B. Two separable failure patterns emerge: (i) a context failure at the Final Report ($P(C_F) = 0.563$, $P(D_F|C_F = 1) = 0.964$) and (ii) a decision failure at the Investigation node ($P(D_I|C_I = 1) = 0.800$). These correspond to Cases A and B respectively, addressed through a diagnostic validation and a deployable context-reduction intervention.

Table 4: Separation validation: Final Report $X_F \to x_F^*$

$P(C_F = 1)$	$56.3^{+10.1}_{-10.7}$
$P(C_F^* = 1)$	$92.9^{+3.8}_{-7.7}$
$P(D_F \mid C_F = 1)$	$96.4^{+1.7}_{-3.2}$
$P(D_F \mid C_F^* = 1)$	$92.7^{+2.2}_{-3.1}$
$P(D_F)$	$58.6^{+4.7}_{-4.8}$
$P(D_F^*)$	$86.1^{+3.0}_{-3.7}$

x_F^*: augmented X_F; D_F^*: decision after replacing X_F with x_F^*.

Table 5: Long context effect on Investigation

Length	N	$P(D_I \mid C_I{=}1)$
≤32K	20	$96.0^{+2.5}_{-6.0}$
>32K	46	$73.0^{+5.4}_{-6.2}$
Total	66	$80.0^{+4.0}_{-4.8}$

Table 6: Decision intervention: Investigation $X_I \to t(X_I)$

$P(D_I \mid C_I = 1)$	$80.0^{+4.0}_{-4.8}$
$P(D_I \mid C_I^* = 1)$	$94.5^{+2.0}_{-3.1}$
$P(D_F)$	$58.5^{+4.7}_{-4.9}$
$P(D_F \mid D_I^*)$	$74.2^{+4.0}_{-4.5}$

D_I^*: decision after replacing X_I with $t(X_I)$.

Granite 3.2 8B. Failures compound aggressively: $P(C_I) = 0.477$ and $P(D_I|C_I = 1) = 0.290$, a Case C joint failure where the mechanism estimate is unreliable due to sparse conditioning.

Separation property verification. The ε_i row confirms the approximate product $P(D_i) \approx \mu_i \cdot P(C_i)$ holds with at most 4.0pp error across all models and nodes (≤6.4pp at the upper CI bound). The noise floor $\lambda_i = P(D_i \mid C_i = 0)$ is itself non-negligible at Bootstrap and Final Report (5–17% for the stronger models), reflecting labeling approximation — some decisions are correct for reasons not fully captured by the binary C_i label. However, the *contribution* $\varepsilon_i = \lambda_i \cdot (1 - P(C_i))$ remains small because high λ_i coincides with high $P(C_i)$. At Investigation, $\lambda_i = 0$ structurally: an entity that was not investigated cannot be diagnosed as root cause. The separation property $\mu_i \gg \lambda_i$ holds throughout (ratios ≥5:1), validating the A/B/C taxonomy.

3.3 Case Studies

We focus on GPT-OSS-120B, where two failure patterns are clearly identified. We present one Class I diagnostic validation confirming the separation between context supply and mechanism quality and one deployable Class I intervention.

3.3.1 Separation Validation: Final Report (Case A)

The Final Report exhibits $P(C_F) = 0.563$ but $P(D_F|C_F = 1) = 0.964$: a context failure. The diagnostic indicates the fix must be upstream, not at the node itself.

Cross-model differential analysis (§2.9) against o4-mini reveals that the stronger model's contexts consistently contain the root cause entity while GPT-OSS-120B's do not — suggesting a Class I intervention. We validate by replacing X_F with an augmented context x_F^*: for each sample where the root cause was absent from context, we used the o4-mini context from the same incident and an LLM to inject the missing root cause entity into the original context, raising $P(C_F)$ from 0.576 to 0.929. Re-generating decisions on the augmented contexts confirms the diagnosis (Table 4). Mechanism quality is unchanged ($P(D_F|C_F^* = 1) = 0.927$ vs. original 0.964), while the overall decision rate rises from 0.586 to 0.861 purely as a consequence of improved context supply. This validates the separation between context supply and mechanism quality for the Final Report node despite not being a deployable intervention. The case study below demonstrates how one could realistically increase the Final Report node context at runtime.

3.3.2 Context Reduction Intervention: Investigation (Case B)

The Investigation node shows $P(D_I|C_I = 1) = 0.800$: a processing failure (Case B). Metric differential analysis (§2.9) reveals that GPT-OSS-120B struggles with contexts exceeding 32K tokens (Table 5): shorter contexts yield a 96% decision rate vs. 73% for longer ones. The information is present but buried in excessive data suggesting a processability issue, not a capability one.

We replace X_I with $t(X_I)$: a deterministic parser removes duplicate and unnecessary observability fields while preserving diagnostic signals. This is a Class I intervention — only the context changes; the mechanism f_I is unmodified. Table 6 shows that reducing contexts raises $P(D_I \mid C_I^* = 1)$ by 14.5 points. Propagating improved decisions forward to regenerate the Final Report raises $P(D_F)$ by 15.8 points, consistent with the cascade mechanism (Theorem 1). This intervention is completely deployable at runtime as the deterministic parser does not rely on any ground truth labels.

4 Related Work

Error taxonomies for agents. Recent work proposes generalizable error taxonomies for agent systems: MAST [11] for multi-agent systems, TRAIL [2, 6] for single- and multi-agent systems, and domain-specific taxonomies for low-code platforms [8] and software engineering [3]. These taxonomies classify errors by type (planning, reasoning, execution) but do not provide a causal mechanism for how errors propagate through a workflow DAG. Our framework is complementary: the SCM structure localizes *where* and *why* a failure occurs, while error taxonomies characterize *what kind* of error it is.

Rubric-based evaluation. Another line of work frames evaluation as multi-dimensional scoring: T-Eval [1] for tool-calling, TaskBench [16] for task automation, and AssetOpsBench [12] for industrial operations. These approaches aggregate scores across dimensions but do not decompose per-node performance into causal factors or distinguish context failures from mechanism failures.

Automatic failure attribution. Applying LLM-as-judge to agent trajectories [17, 7] remains difficult: it only achieves roughly 50% accuracy at agent-level and under 20% at step-level attribution. DAG-based workflow comparison [14] is more robust but requires diverse ground-truth workflows. Our approach avoids both limitations: it requires only a terminal correctness criterion J and derives intermediate labels via backward propagation through the DAG structure.

Workflow reliability systems. Sherlock [15] identifies error-prone nodes via counterfactual analysis and applies selective verification with speculative downstream execution, recovering from failures at runtime. These systems optimize for reliability under a fixed workflow structure; our framework instead diagnoses *where* and *why* a workflow underperforms, informing structural interventions.

Causal models. The structural causal model formalism [13] provides the formal backbone. Our contribution is applying SCMs to LLM workflow debugging: the key novelty is the informational definition of instrumental correctness combined with the separation property, which yields the node decomposition and exhaustive taxonomy that standard causal diagnostics do not provide out of the box.

5 Conclusion

We addressed the problem of localizing failures in multi-node LLM agent workflows when end-to-end accuracy does cannot identify which node is responsible and whether the bottleneck is upstream context or local decision-making. Our contribution is a structural causal framework that formalizes disambiguating context failures from mechanism failures at every node, prescribing the appropriate intervention class, and tracing failures backward through the DAG to a root cause. Validation on IT-Bench across three models provides an initial demonstration that the framework correctly localizes distinct failure modes; a targeted intervention yields a 15pp absolute gain (27% relative) end-to-end.

Limitations and future work. The framework assumes a static DAG and extending to agents with dynamic branching requires normalizing structural variation. The binary correctness label is a simplification that discards partial-correctness signal. The separation property $\mu_i \gg \lambda_i$ is an empirical precondition, not a guarantee. For tasks where models can succeed without context (high λ_i), the decomposition remains valid but loses diagnostic power. Semi-automatic labeling schemes are necessary to scale the methodology, and their validation is still non-trivial. As the experiments are conducted on a single agent in a single domain, generalizability across architectures and tasks is the primary open question. Future work will extend the framework to dynamic topologies, investigate scalable labeling procedures including cost analysis, and validate across diverse agent architectures and domains.

References

[1] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In Lun-Wei Ku, Andre Martins,

and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[2] Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization, 2025.

[3] Shubham Gandhi, Jason Tsay, Jatin Ganhotra, Kiran Kate, and Yara Rizk. When agents go astray: Course-correcting SWE agents with PRMs. In *Workshop on Scaling Environments for Agents*, 2025.

[4] IBM. Granite 3.2-8B-Instruct Model Card, 2025.

[5] Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Jae-wook Ahn, Debanjana Kar, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas Fuller, and Ruchir Puri. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[6] NVJK Kartik, Garvit Sapra, Rishav Hada, and Nikhil Pareek. Agentcompass: Towards reliable evaluation of agentic workflows in production, 2025.

[7] 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 *The Fourteenth International Conference on Learning Representations*, 2026.

[8] Xuyan Ma, Xiaofei Xie, Yawen Wang, Junjie Wang, Boyu Wu, Mingyang Li, and Qing Wang. Diagnosing failure root causes in platform-orchestrated agentic systems: Dataset, taxonomy, and benchmark, 2025.

[9] OpenAI. gpt-oss-120b & gpt-oss-20b Model Card, 2025.

[10] OpenAI. o3 and o4-mini System Card, 2025.

[11] Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025.

[12] Dhaval Patel, Shuxin Lin, James Rayfield, Nianjun Zhou, Roman Vaculin, Natalia Martinez, Fearghal O'donncha, and Jayant Kalagnanam. Assetopsbench: Benchmarking ai agents for task automation in industrial asset operations and maintenance, 2025.

[13] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

[14] Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[15] Yeonju Ro, Haoran Qiu, Íñigo Goiri, Rodrigo Fonseca, Ricardo Bianchini, Aditya Akella, Zhangyang Wang, Mattan Erez, and Esha Choukse. Sherlock: Reliable and efficient agentic workflow execution, 2025.

[16] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

[17] Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025.

A Agent Node Decomposition

The original agentic workflow was decomposed into five LLM call-based nodes and two "virtual" node types that make up the most important steps of the system. The virtual node types replicate system behavior that aggregates results from previous nodes and does not require calls to an LLM. Figure 2 shows the full workflow DAG.

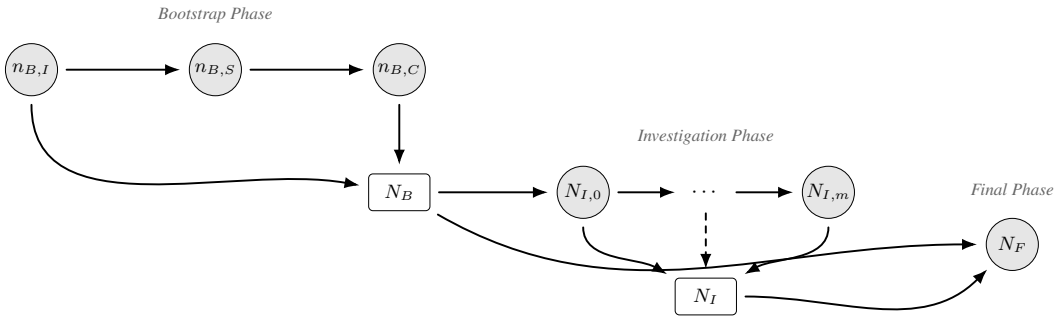

Figure 2: Agent workflow DAG. Circles denote LLM nodes; rectangles denote virtual (non-LLM) aggregator nodes. $n_{B,I}$: Initial Entity Selection; $n_{B,S}$: Initial Investigation Summarization; $n_{B,C}$: Change Event Analysis; N_B: Bootstrap aggregator; $N_{I,0} \to N_{I,m}$: Individual Entity Judgment nodes forming a sequential chain; each node also feeds into N_I (dashed edge indicates all intermediate nodes do likewise); N_I: Investigation aggregator; N_F: Diagnosis Final Report.

Bootstrap Phase **LLM Call: Initial Entity Selection.** Based on observability and alert data, the agent produces a structured JSON of one to three most likely primary failure candidate entities. Evaluated against silver ground truth.

LLM Call: Initial Investigation Summarization. The agent generates a one-page summarization of the findings from the previous step, including the candidate entity names. Evaluated against silver ground truth.

LLM Call: Change Event Analysis. Given a list of recent change events along with the summary of the initial investigation, the agent generates a list of change events that were likely contributors to the incident under investigation. Evaluated against silver ground truth.

Virtual Node: Bootstrap ($i = B$). Collects all entities chosen by Initial Entity Selection, Change Event Analysis, and Entity Judgment. For evaluation, each entity is assigned a silver label of root cause or non-root cause. Evaluation considers whether a root cause entity appears in the collected set.

Investigation Phase **LLM Call: Individual Entity Judgment.** Using observability and alert data for a given entity along with a history of the ongoing investigation, the agent classifies the entity as a primary failure, symptom, or exonerated entity.

Virtual Node: Investigation ($i = I$). Aggregates the judgments of all seen entities. Evaluation considers whether at least one root cause entity is classified correctly.

Final Phase **LLM Call: Diagnosis Final Report ($i = F$).** The agent considers all entity judgments and the investigation history to produce a list of entities it has diagnosed as the primary failure cause for the incident. Evaluated against gold ground truth.

B Node-Level Correctness Labeling

The end-to-end ground truth file for each incident was authored by a Kubernetes subject matter expert and contains detailed information about the fault injected into the system. Most importantly, each incident specifies a root cause entity and its full error propagation path. Each entity is described by name, kind, and associated filters. These ground truth artifacts are the basis for deriving node-level instrumental correctness labels at each of the three virtual or terminal nodes.

Bootstrap ($i = B$). The bootstrap node is labeled correct if at least one of the entities it selected lies on the ground truth error propagation path. However, a subset of incidents are exceptions: for some incidents, entities on the downstream propagation path are not topological neighbors in the system graph, meaning the agent would never have been able to reach the root cause entity by traversal alone. For these incidents, the stricter criterion applies: the bootstrap node is labeled correct only if at least one selected entity is the ground truth root cause entity itself. In both cases, entity matching is performed using an LLM-as-judge, which compares each selected entity's name, kind, and filters against the ground truth entries.

Investigation ($i = I$). Each entity judgment produced during the investigation is matched against the ground truth root cause entity using the ground truth file. Because the raw investigation output contains reasoning and other contextual information in addition to the judgment label, an LLM-as-judge is used to parse the judgment classification (primary failure, symptom, or exonerated) from the full output. The investigation node is an aggregate of all entity judgments seen during that invocation. It is labeled correct if at least one root cause entity that was investigated received a judgment of `PRIMARY FAILURE`.

Final Report ($i = F$). The original system end-to-end evaluation prompt is used directly. This prompt was developed as part of the IT-Bench evaluation harness and assesses whether the final diagnosis correctly identifies the root cause entity against the gold ground truth.

C Case Study Data Statistics

Table 7: Case study context-decision pairs across node types and models

Node Type	o4-mini	GPT-OSS-120B	Granite 3.2 8B
Initial Entity Selection	88	87	90
Initial Investigation Summarization	88	87	90
Change Event Analysis	88	87	90
Entity Judgment	510	664	1192
Diagnosis Final Report	88	85	85