# AVERT: A TOOL-VERIFIED AGENT WITH RESPONSIBILITY-ROUTED RULE EVOLUTION FOR TELECOM ROOT CAUSE ANALYSIS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Telecom root cause analysis (RCA) is critical for service restoration and repair planning during multi-site outages. LLM agents struggle because evidence is scattered across long alarm streams and topology graphs, candidate causes may lack evidential support, and free-form reasoning is difficult to verify. To address these challenges, we present AVERT, an RCA agent that follows an iterative hypothesis–diagnosis–verification workflow. Given the incident evidence, AVERT first forms an initial root-cause hypothesis. The Rule-Constraint Layer then structures the diagnosis into three stages: candidate enumeration, causal and topological constraint checking, and verdict selection, producing a provisional diagnosis. The Verification-Feedback Layer tests this diagnosis against service-path impact and feeds blocked paths, surviving sites, and counterexamples back to revise the hypothesis and rerun the rule stages. To keep the rule layer adaptive as incident patterns evolve, we introduce offline Responsibility-Routed Rule Evolution, which uses evaluator feedback and execution traces to identify the responsible stage and update only the corresponding rules. We evaluate AVERT on TelcoOutageBench, comprising 131 real-world telecom incidents. Compared with frontier model baselines, AVERT improves the mean RCA Score by 2.71–13.81 percentage points across all eight backbones, including Claude Opus 4.6, GPT-5.4, and GLM-5.1.

## 1 INTRODUCTION

Root cause analysis (RCA) in telecom network operations aims to rapidly localize faulty sites and network elements requiring repair, and to identify the corresponding fault types responsible for service outages (Steinder & Sethi, 2004; Zhang et al., 2021; Jiang & Bai, 2023). When a group outage affects multiple wireless sites, operators must determine the underlying causes promptly to restore service. RCA is an evidence-driven task: a typical incident provides an incident ticket specifying outage time and affected sites, a topology graph describing connectivity among network elements, and alarm tables with timestamps, sources, locations, and severities. The goal is to infer all independent root causes that require repair, each represented as a structured triple of faulty sites, involved network elements, and a cause type, such as power failure, fiber failure, or hardware failure. Unlike single-label classification, telecom RCA is a multi-causal diagnostic problem: accurate solutions must first localize evidence-supported faulty sites and elements, and then determine the failure category to guide repair strategies.

Although recent LLM agents have shown strong reasoning-and-acting and software-engineering capabilities (Yao et al., 2023; Yang et al., 2024b), directly applying them to telecom RCA remains difficult. In real-world telecom operations, RCA is still largely performed manually by engineers who inspect incident tickets, alarm logs, topology information, and operational experience. This manual process is time-consuming and evidence-intensive (Pei et al., 2025), and difficult to scale as networks become increasingly heterogeneous and fault patterns continue to evolve. Recent studies suggest that LLM agents can support operational RCA by collecting diagnostic evidence beyond the initial incident report (Roy et al., 2024), but direct prompting faces three key challenges. **Long-context evidence overload:** telecom incidents contain long and noisy alarm tables, where sparse root-cause evidence is easily buried among derivative or irrelevant alarms (Liu et al., 2024). **Missing closed-loop verification:** generated diagnoses are typically not checked against whether the



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.

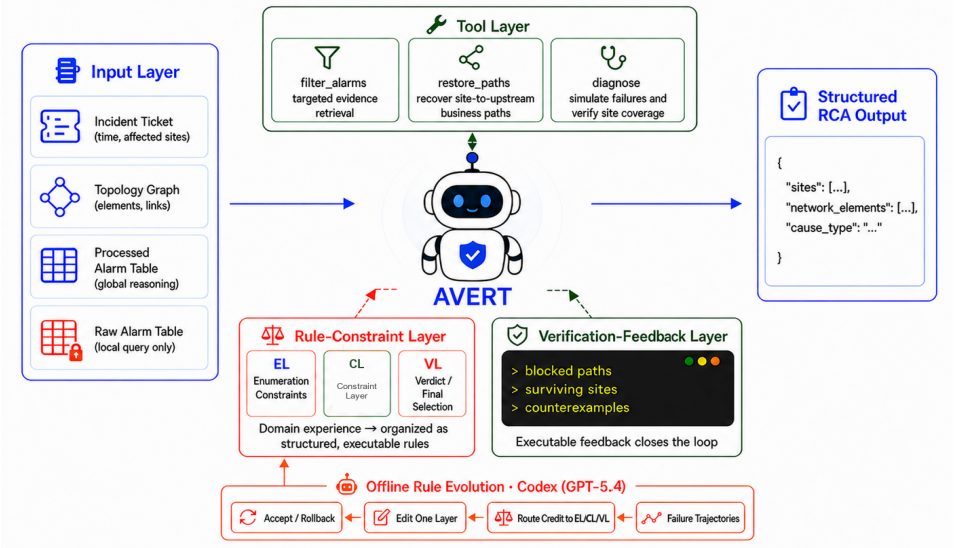



Figure 1: AVERT couples tool-verified incident-time diagnosis with offline Responsibility-Routed Rule Evolution. The EL/CL/VL policy separates candidate enumeration, causal validation, and final selection; evidence tools expose alarms and business paths, while the verifier reports the simulated impact of a hypothesized failure on affected sites and paths. During offline evolution, Codex (GPT-5.4) combines evaluator error profiles with execution traces to identify the responsible rule layer and perform a localized update.

predicted faulty sites or network elements would actually disconnect all affected service paths. **Distorted causal reasoning:** free-form reasoning can treat downstream alarms as root causes, infer a fault type without direct evidence, or stop once a partial explanation covers only a subset of affected sites. These limitations suggest that telecom RCA cannot be solved by longer context or stronger prompting alone; the model needs structured evidence access, topology-level verification, and domain constraints that keep reasoning aligned with operational evidence.

To address these challenges, we introduce AVERT, a telecom-specific LLM agent that forms a closed diagnostic loop (Figure 1). AVERT combines a Tool Layer for targeted alarm retrieval, service-path restoration, and topology-impact simulation with a Rule-Constraint Layer that organizes domain knowledge into EL, CL, and VL rules for candidate enumeration, causal and topological validation, and final selection. Once a candidate root cause is generated, the Verification-Feedback Layer evaluates it against the restored service paths and feeds blocked paths, surviving sites, and counterexamples back into hypothesis revision.

A fixed Rule-Constraint Layer may become suboptimal as incident patterns evolve. We therefore introduce offline Responsibility-Routed Rule Evolution, which combines evaluator feedback with failure trajectories to localize the first responsible layer and route a targeted update to EL, CL, or VL. On TelcoOutageBench, AVERT improves mean RCA Score by 2.71–13.81 points over direct prompting across all eight backbones; on Qwen3.5-27B, Responsibility-Routed Rule Evolution further adds 6.16 points beyond the tool-verified agent.

**Our contributions are threefold:**

- **Formalization and verifiable evaluation.** We formalize multi-cause group-outage RCA as set-valued structured prediction: affected sites follow a topology-impact criterion, independent root causes are failures not causally downstream of one another, and predictions are scored by a matching-aware evaluator that couples Hungarian root-cause correspondence with repair-relevant field accuracy.
- **AVERT.** We introduce a verifier-in-the-loop RCA agent that conditions evidence access on the current hypothesis and tests, through deterministic topology-impact simulation, whether a hypothesized root cause reproduces the observed outage, turning each diagnosis into a falsifiable, evidence-grounded claim.

- **EL/CL/VL with Responsibility-Routed Rule Evolution.** We parameterize the diagnostic policy as three rule layers and route evaluator-and-trace feedback to the single responsible layer, replacing monolithic prompt rewriting with local credit assignment. Responsibility-routed evolution achieves stronger and more stable RCA performance than unrestricted or monolithic rule updates.

## 2 Related Work

**Network RCA and operational agents.** Network fault localization has progressed from alarm-correlation and fault-propagation models (Katzela & Schwartz, 1995; Steinder & Sethi, 2004) to path-, dependency-, and causal-graph diagnosis in enterprise, cloud, microservice, and telecom systems (Chen et al., 2004; Kandula et al., 2009; Wang et al., 2018; Wu et al., 2020; Zhang et al., 2021; Jiang & Bai, 2023). Recent operational agents acquire evidence interactively, expose cloud environments through agent interfaces, or constrain diagnosis with operational procedures (Roy et al., 2024; Shetty et al., 2024; Pei et al., 2025; Wu et al., 2025). AVERT studies real group-outage incidents that may contain multiple independent causes, requiring actionable sites, network elements, and cause types whose predicted failures reproduce the observed service-path impact.

**Tool-augmented and executable reasoning.** Language agents interleave reasoning with external actions or learn to select APIs through general and task-specific interfaces (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Patil et al., 2024; Yang et al., 2024b). Execution-grounded methods further test model reasoning through external programs (Gao et al., 2023; Lyu et al., 2023). AVERT adapts these ideas through telecom-specific alarm retrieval and service-path restoration, together with deterministic simulation of predicted node and link failures. Its tools therefore provide both targeted evidence and a physical test of whether a hypothesis can explain the outage.

**Feedback-driven agent optimization.** Prompt optimizers use model-guided search, textual gradients, or evolution to improve instructions (Zhou et al., 2023; Yang et al., 2024a; Pryzant et al., 2023; Wang et al., 2024; Guo et al., 2024; Fernando et al., 2024). DSPy, MIPRO, and TextGrad extend optimization to compound language-model programs (Khattab et al., 2024; Opsahl-Ong et al., 2024; Yuksekgonul et al., 2024); feedback-based agents reuse critiques or experience across attempts (Madaan et al., 2023; Shinn et al., 2023; Zhao et al., 2024). Recent systems further optimize tool-use policies or complete agent workflows (Wu et al., 2024; Zhang et al., 2025). AVERT structures its optimization target as EL/CL/VL and uses execution trajectories to route evaluator-guided updates to one responsible layer, preserving rules for other diagnostic stages and reducing interference during Rule Evolution.

## 3 Problem and Benchmark

### 3.1 Task Definition

We study root cause analysis for telecom group-outage incidents. Each incident is represented by heterogeneous operational evidence $x = (q, G, A)$, where $q$ is an incident ticket, $G = (V, E)$ is a network topology graph over network elements, and $A$ is a set of alarm records. The ticket specifies the outage time and a set of affected wireless sites, which describe the observed service impact rather than the repair location. The topology graph captures physical connectivity among network elements. Alarm records provide timestamped observations with source, site, location, and severity metadata.

In $G = (V, E)$, each vertex $v \in V$ is a network element, such as a router, base-station-side device, or other transport equipment that can appear as an alarm source or topology node. Each edge $e = (u, v) \in E$ denotes a physical connection between two network elements. We distinguish *sites* from *network elements*: a site is a physical or operational location identified by a site ID, while network elements are devices located at, or associated with, a site. A single site can contain multiple network elements.

For a site $s$, let $N(s) \subseteq V$ denote the network elements associated with that site. A *business path* of $s$ is a connected sequence of network elements that starts from an element in $N(s)$ and reaches an upstream aggregation or core element, representing a route through which the site's service traffic

can be carried. A site is affected by an incident if and only if all business paths of all network elements in $N(s)$ are cut by failed nodes or failed edges. Conversely, if at least one business path of at least one site element remains available, the site is not disconnected under this topology-impact semantics.

Given an incident $x$, the task is to predict all independent root causes that should be repaired. Here, independent root causes refer to failures that are not causally downstream of one another; instead, they jointly produce the observed group outage. The output is a set of structured root-cause triples, $r_i = (\texttt{sites}_i, \texttt{network_elements}_i, \texttt{cause_type}_i)$, where `sites` denotes the evidence-supported faulty site or sites, `network_elements` denotes the faulty network element or link endpoints when applicable, and `cause_type` specifies the repair category. In our setting, the cause type is selected from three operational categories: power failure, fiber failure, and hardware failure. For site-level power failures, the repair target is the site power system itself, so the network-element field can be empty.

Each predicted triple must be supported by alarm and topology evidence and specify the repair location, involved network elements or link endpoints, and intervention required to restore service.

### 3.2 TelcoOutageBench

We curate TelcoOutageBench from 131 real-world anonymized group-outage work orders from the production network operations of a telecommunications industry partner.

Each incident exposes three initial input fields to the model: the incident ticket, the required topology data, and the processed alarm table. Raw alarms remain accessible only through the alarm-query tool because the full tables are too large and noisy for direct prompting; an incident contains a median of 308 raw alarm records and as many as 6,615. Appendix A.1 details this constraint and its experimental interpretation.

Ground-truth root causes are annotated by professional telecom operations engineers. Each annotated root cause contains the faulty sites, involved network elements, cause type, and a raw textual description of the repair reason. When an incident has multiple root causes, the annotations separate failures that are not causally downstream of one another but jointly lead to the observed group outage, following the independence definition in Section 3.1.

Table 1: Statistics of the 131-incident TelcoOutageBench.

| Statistic | Min | Mean | Median | Max |
|---|---|---|---|---|
| Affected wireless sites per incident | 4 | 6.40 | 5 | 30 |
| Topology edges per incident | 13 | 49.60 | 39 | 208 |
| Processed alarms per incident | 4 | 75.16 | 59 | 506 |
| Raw alarms per incident | 65 | 718.71 | 308 | 6615 |
| Root causes per incident | 1 | 1.85 | 2 | 6 |

Across its 131 incidents, TelcoOutageBench contains 242 annotated root causes: 113 power failures (46.7%), 107 fiber failures (44.2%), and 22 hardware failures (9.1%); incidents contain a median of two independent root causes.

### 3.3 Evaluation Metrics

Each incident may contain multiple independent root causes, requiring evaluation to account for both root-cause coverage and attribute accuracy. Let $\hat{\mathcal{R}} = \{\hat{r}_i\}_{i=1}^{n_p}$ and $\mathcal{R} = \{r_j\}_{j=1}^{n_g}$ denote the predicted and ground-truth root-cause collections. We define the affinity between two root causes by the F1 overlap of their site sets and obtain correspondence through maximum-weight bipartite matching:

$$a_{ij} = F1(\hat{r}_i.\texttt{sites}, r_j.\texttt{sites}),$$
$$\pi^\star = \arg\max_{\pi \in \Pi} \sum_{(i,j)\in\pi} a_{ij}, \qquad \mathcal{M} = \{(i,j) \in \pi^\star \mid a_{ij} > 0\}.$$

where $\Pi$ is the set of one-to-one assignments and $\pi^\star$ is computed with the Hungarian algorithm (Kuhn, 1955). Writing $m = |\mathcal{M}|$, we define match precision, match recall, and their harmonic mean, Match F1, as

$$P_{\text{match}} = \frac{m}{n_p}, \qquad R_{\text{match}} = \frac{m}{n_g}, \qquad F1_{\text{match}} = \frac{2P_{\text{match}}R_{\text{match}}}{P_{\text{match}} + R_{\text{match}}}.$$

For each matched pair $(i, j) \in \mathcal{M}$, we evaluate the sites, network elements, and cause type independently. Let $\hat{S}_k^i$ and $S_k^j$ denote the predicted and ground-truth values of field $k$. The pairwise field score is the standard set F1,

$$P_k^{ij} = \frac{|\hat{S}_k^i \cap S_k^j|}{|\hat{S}_k^i|}, \qquad R_k^{ij} = \frac{|\hat{S}_k^i \cap S_k^j|}{|S_k^j|}, \qquad F1_k^{ij} = \frac{2P_k^{ij}R_k^{ij}}{P_k^{ij} + R_k^{ij}}.$$

For set-valued fields, an empty prediction and empty ground truth are treated as an exact match, while a one-sided empty set receives zero. Cause type is represented as a singleton set and therefore requires an exact label match. Field-level performance is macro-averaged across matched root causes:

$$\overline{F1}_k = \frac{1}{m} \sum_{(i,j)\in\mathcal{M}} F1_k^{ij}.$$

We refer to the resulting macro-averages for sites, network elements, and cause type as Sites F1, NE F1, and Type F1, respectively. For a matched ground-truth power failure whose network-element annotation is empty, the network-element field is not applicable: its 0.3 weight is reassigned to the site field. Thus the matched-root Field Score is

$$f_{\text{field}}^{ij} = \begin{cases} 0.8\, F1_{\text{sites}}^{ij} + 0.2\, F1_{\text{type}}^{ij}, & \text{power failure with } S_{\text{NE}}^j = \emptyset, \\ 0.5\, F1_{\text{sites}}^{ij} + 0.3\, F1_{\text{NE}}^{ij} + 0.2\, F1_{\text{type}}^{ij}, & \text{otherwise,} \end{cases}$$

$$F_{\text{field}} = \frac{1}{m} \sum_{(i,j)\in\mathcal{M}} f_{\text{field}}^{ij},$$

$$S_{\text{RCA}} = F1_{\text{match}}\, F_{\text{field}}.$$

The resulting RCA Score, denoted by $S_{\text{RCA}}$, is high only when a prediction recovers the incident's distinct root causes and accurately specifies their repair-relevant attributes. We report incident-level macro-averages of Match F1, Field Score, Sites F1, NE F1, Type F1, and RCA Score.

# 4 AVERT

AVERT is implemented as an agent harness inspired by mini-SWE-Agent and the broader SWE-agent line of work on agent-computer interfaces (Yang et al., 2024b). It separates model inference, the execution environment, and the agent loop while adapting them to telecom RCA. The agent first forms an initial root-cause hypothesis from the incident ticket, topology, and processed alarms, then enters a tool-assisted revision stage in which raw alarms and path evidence are retrieved on demand. A common CLI exposes the evidence and verification capabilities, and Figure 1 provides an overview of the resulting workflow.

## 4.1 Tool and Verification-Feedback Layers

Following Figure 1, this subsection covers the two coupled incident-time modules: the Tool Layer exposes `filter_alarms`, `restore_paths`, and `diagnose`, while the Verification-Feedback Layer turns verification outputs into evidence for hypothesis revision.

Telecom incidents combine long, noisy alarm tables with topology data that does not directly expose service routes. AVERT conditions evidence access on the current hypothesis: the `filter_alarms` tool retrieves alarms associated with implicated sites and network elements, while `restore_paths` recovers site-to-upstream paths and shared convergence points. Together, these tools narrow the alarm and topology search spaces before verification.

The diagnosis verifier maps predicted causes to failed nodes and links and simulates their impact on the restored service paths. It returns covered sites and surviving paths so that the agent can revise

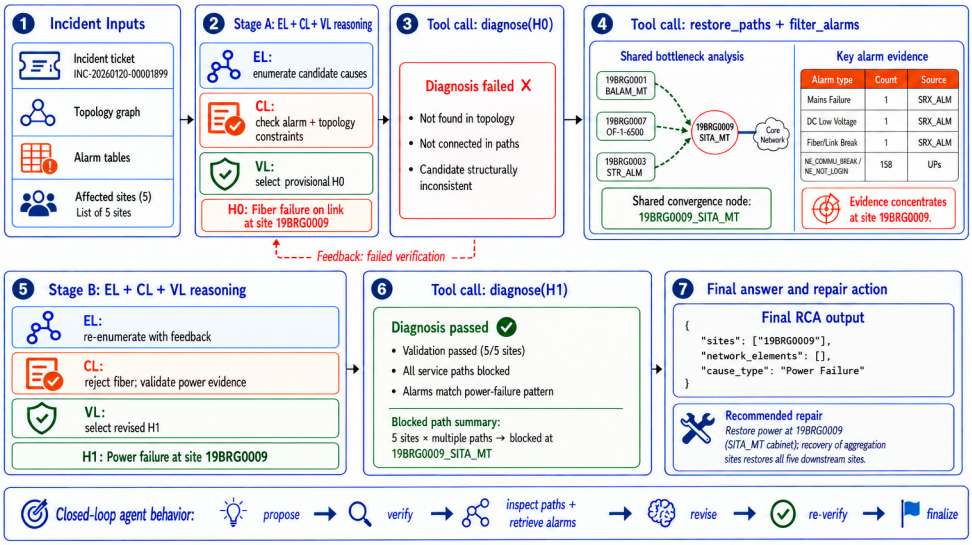



Figure 2: Verifier-in-the-loop diagnosis on a representative telecom incident. AVERT first proposes a locally plausible fiber-link failure, but topology verification exposes that the candidate is structurally inconsistent. Restored service paths and targeted alarm retrieval reveal a shared convergence site with earlier power evidence, leading the agent to revise and re-verify its hypothesis before producing an actionable root cause.

candidates that leave part of the outage unexplained. A site is topologically covered only when the predicted failures block all of its available paths; evidence support and causal admissibility remain governed by the EL/CL/VL policy. Figure 2 illustrates this interaction on a representative group-outage incident.

### 4.2 Rule-Constraint Layer: Structured EL/CL/VL Diagnosis

AVERT represents its diagnostic policy as an ordered collection of three rule sets: the Enumeration Layer (EL), the Constraint Layer (CL), and the Verdict Layer (VL). These components follow the causal workflow of an experienced operator: construct possible causes, test whether each cause is admissible, and consolidate the surviving explanations. The decomposition makes the role of each rule explicit and prevents recall-oriented guidance, causal filtering, and final-set selection from being conflated. The same EL/CL/VL policy governs both initial hypothesis formation and subsequent tool-assisted revision. Its staged structure also exposes where a diagnostic failure enters the computation, which later enables layer-specific evolution. Gray boxes quote representative evolved rules from Appendix C.3.

$$x \xrightarrow{\mathrm{EL}} \mathcal{C}_0 \xrightarrow{\mathrm{CL}} \mathcal{C}_1 \xrightarrow{\mathrm{VL}} \hat{\mathcal{R}}.$$

The **Enumeration Layer (EL)** constructs a high-recall candidate set. Its rules map direct alarms, link symptoms, shared upstream elements, and temporally separated failures into candidate triples containing `sites`, `network_elements`, and `cause_type`. EL also preserves independently supported causes that may jointly produce the outage. Applied to incident evidence $x$, EL produces an initial candidate set $\mathcal{C}_0$, giving subsequent reasoning access to plausible repair hypotheses before the model commits to a compact answer.

> **EL-1 (enumeration).** If a network element or link has an alarm that directly indicates a root cause, construct a candidate.

The **Constraint Layer (CL)** encodes reusable admissibility conditions grounded in network mechanisms. Its rules test temporal precedence, upstream coverage, path redundancy, alarm directness, and consistency between evidence and output fields. By applying these conditions to $\mathcal{C}_0$, CL produces

a surviving candidate set $\mathcal{C}_1$ and gives the agent an explicit boundary between an alarm-correlated hypothesis and a causally admissible root cause.

> **CL-2 (elimination).** If a candidate is neither on any upstream path of an affected site nor capable of explaining the shared failure pattern among affected sites, eliminate it.

The **Verdict Layer (VL)** converts $\mathcal{C}_1$ into the final root-cause set $\hat{\mathcal{R}}(x)$. VL ranks candidates by evidence strength and explanatory scope, merges duplicate descriptions of the same physical failure, removes downstream manifestations, and preserves multiple causes when independent evidence supports them. Together, EL, CL, and VL separate three error modes that remain entangled when all diagnostic instructions are optimized as one monolithic policy: missed causes, causally invalid causes, and poorly consolidated verdicts. Beyond guiding incident-time reasoning, this decomposition provides an explicit unit of credit for Rule Evolution. It therefore serves as both a structured diagnostic prior and a modular parameterization of the policy, allowing incident-level feedback to improve one decision stage without rewriting rules that govern the others.

> **VL-1 (ranking).** A candidate with direct evidence has higher priority than a candidate supported only by an indirect link symptom.

### 4.3 Offline Responsibility-Routed Rule Evolution

The structured policy creates a natural search space for data-driven improvement. We use Codex (GPT-5.4) as the offline evolution driver, while the incident-level RCA policy continues to run with the Qwen3.5-27B backbone. Let $\mathcal{R}^{(t)}$ be the incumbent rule set at evolution round $t$. AVERT executes this policy on historical incidents, and the evaluator scores root-cause matching together with field-level precision and recall for every incident. The incidents with the lowest RCA Scores form a feedback batch $\mathcal{B}^{(t)}$. For each incident, Codex receives the evaluator breakdown and the corresponding execution trace, including model revisions, tool calls, observations, and the final structured answer. These two signals have complementary roles: evaluator feedback characterizes how the final diagnosis failed, while the execution trace reveals where that failure first entered the staged policy. Figure 3 summarizes the resulting local-update loop.

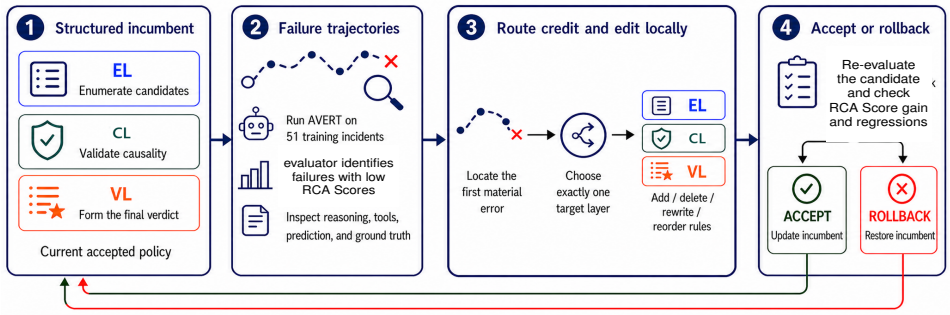



Figure 3: Responsibility-Routed Rule Evolution driven by Codex (GPT-5.4). The incumbent EL/-CL/VL policy is evaluated on 51 evolution incidents; evaluator feedback and the corresponding trajectory identify the first material error and route the update to one responsible rule layer. The other layers remain fixed, and the candidate is accepted only when RCA Score and regression gates pass. Otherwise, rollback preserves the incumbent.

Routing combines evaluator feedback with execution traces. Field-level precision and recall deficits provide a coarse error profile, and Codex uses the corresponding trace to localize the first responsible decision: candidate omissions are assigned to EL, invalid filtering decisions to CL, and post-filter consolidation errors to VL. For round $t$, Codex selects exactly one responsible layer $k_t \in \{\mathrm{EL}, \mathrm{CL}, \mathrm{VL}\}$ and proposes a structured rule edit

$$\widetilde{\mathcal{R}}_{k_t}^{(t+1)} = U\left(\mathcal{R}_{k_t}^{(t)}, \mathcal{B}^{(t)}\right), \qquad \widetilde{\mathcal{R}}_{\ell}^{(t+1)} = \mathcal{R}_{\ell}^{(t)} \text{ for } \ell \neq k_t,$$

where $k_t$ denotes the responsible layer selected at round $t$, $\ell$ indexes either non-target layer, and $\mathcal{B}^{(t)}$ is the evaluator-and-trace feedback batch. The update operator may add, rewrite, delete, or reorder rules within the selected layer, while the other two layers remain unchanged. This constraint turns responsibility attribution into a localized policy update and prevents an error assigned to one diagnostic stage from perturbing rules at another.

Each candidate is executed on the full evolution set. We retain it when its average RCA Score improves beyond a minimum margin and the fraction of incidents with a material regression remains below a threshold. A rejected candidate is rolled back to $\mathcal{R}^{(t)}$, and the most regressed incumbent–candidate trajectory pairs are returned as contrastive evidence for the next round. An accepted candidate becomes $\mathcal{R}^{(t+1)}$. Precise routing to the responsible layer is central to the method: evaluator profiles and execution traces localize each failure to the rule subset that produced it, allowing the correction to preserve policies governing other diagnostic stages. Aggregate improvement and incident-level regression gates then act as safeguards against destructive updates. The resulting procedure combines local credit assignment with regression control, reducing cross-layer interference while preserving rules that already support correct behavior.

# 5 Experiments

## 5.1 Setup

**Evaluation protocol.** We evaluate all methods on the 80-incident test split using the metrics of Section 3.3. All receive the incident ticket, topology graph, and processed alarms. Table 2 is a three-run, backbone-matched end-to-end comparison; Appendix A.1 discusses the raw-alarm access constraint.

**Baselines and AVERT.** We compare Direct Prompt and AVERT on eight backbones with identical initial evidence, decoding budget, and output schema within each pair. Direct Prompt has no RCA-specific tools; AVERT adds evidence retrieval, topology verification, and EL/CL/VL. We also report a deterministic traditional graph/rule baseline that uses processed alarms and topology without an LLM; its single-run point estimate is included in Table 2. Backbone matching controls model strength, while Table 3 separates the cumulative components.

**Rule Evolution protocol.** We evolve the policy on the earlier 51-incident training split for 15 rounds using Codex (GPT-5.4). Experiment 1 is the no-evolution baseline; Experiments 2–4 add a monolithic policy, editable EL/CL/VL, and responsibility-routed EL/CL/VL updates, respectively. Acceptance, rollback, and final-prompt selection use only the 51 training incidents; the test split supplies no feedback or gates. Experiment 4's final accepted prompt is fixed at round 11, and all per-round test scores are retrospective. The complete evolution protocol, including acceptance and rollback, is provided in Appendix E; complete model and serving settings are in Appendix F; and detailed ablation results are in Appendix A.

## 5.2 Main Results

Table 2 reports the backbone-matched comparison together with the deterministic traditional graph/rule reference, while Table 3 isolates cumulative components under Qwen3.5-27B. Complete four-stage ablations for every backbone are reported in Appendix A.

**End-to-end and component results.** AVERT improves RCA Score on all eight backbones by +2.71 to +13.81 points. The traditional graph/rule baseline reaches a 57.45% RCA Score and 63.67% Match F1, providing a deterministic non-LLM reference; Direct Prompt varies around this reference, whereas every AVERT configuration exceeds it. These gains control backbone choice but include tool-mediated evidence access. Table 3 separates Tools, Verifier, and Rule Evolution; Appendix A.1 gives the full interpretation and Table 4 extends the ablation to every backbone.

**Cross-backbone portability of evolved rules.** The EL/CL/VL policy was evolved once with Qwen3.5-27B and applied verbatim to every backbone. Relative to the +Verifier stage in Table 4, the evolved rules raise RCA Score on six of eight backbones (+6.16 on the source model, +4.11 on Gemma4-31B, +1.43 to +1.99 on Qwen3.5-9B, Gemma4-12B, GLM-5.1, and GPT-5.4) and slightly lower it on DeepSeek-v3.2 ($-0.61$) and Claude Opus-4.6 ($-1.82$); rules distilled from one model's

Table 2: Backbone-matched comparison on the 80-incident test split. Entries are percentages; LLM rows report three-run mean±standard deviation and the traditional graph/rule row is a single run. $\Delta S_{\mathrm{RCA}}$ is the AVERT minus Direct Prompt RCA Score gain.

| Backbone | Configuration | Metrics (%) | | | | | | $\Delta S_{\mathrm{RCA}}$ |
|---|---|---|---|---|---|---|---|---|
| | | Match F1 | Field Score | Sites F1 | NE F1 | Type F1 | RCA Score | |
| Qwen3.5-9B | *Direct* | 74.09±1.31 | 66.93±1.62 | 59.72±1.15 | 78.58±2.79 | 67.50±2.16 | 51.13±0.56 | – |
| | **AVERT** | 69.35±3.58 | 71.92±3.32 | 71.10±1.97 | 80.10±1.67 | 61.73±3.73 | 57.81±1.28 | +6.68 |
| Qwen3.5-27B | *Direct* | 78.63±0.39 | 81.82±0.33 | 85.33±3.88 | 87.72±2.57 | 64.20±1.37 | 67.60±1.47 | – |
| | **AVERT** | 90.23±2.50 | 89.29±3.22 | 91.36±0.77 | 95.16±3.00 | 75.31±0.62 | 81.41±1.57 | +13.81 |
| Gemma4-12B | *Direct* | 72.88±2.50 | 72.34±2.06 | 75.34±2.22 | 75.66±3.85 | 59.88±3.79 | 54.95±0.95 | – |
| | **AVERT** | 77.10±2.67 | 77.38±0.70 | 81.98±1.66 | 84.26±3.46 | 55.56±2.12 | 60.97±2.64 | +6.02 |
| Gemma4-31B | *Direct* | 83.01±3.78 | 77.97±0.38 | 81.92±0.62 | 84.63±3.43 | 58.13±1.19 | 66.26±1.69 | – |
| | **AVERT** | 86.90±2.64 | 87.85±0.24 | 91.19±1.90 | 92.28±0.45 | 72.84±2.44 | 77.42±1.08 | +11.16 |
| DeepSeek-v3.2 | *Direct* | 76.28±2.02 | 83.37±3.30 | 86.14±2.54 | 89.20±3.04 | 67.72±0.16 | 66.62±2.09 | – |
| | **AVERT** | 79.95±0.33 | 86.61±2.68 | 89.39±2.74 | 94.12±3.77 | 68.38±1.45 | 69.87±2.27 | +3.25 |
| GLM-5.1 | *Direct* | 83.19±3.05 | 87.11±1.37 | 89.76±3.64 | 91.65±2.94 | 73.66±3.88 | 74.41±1.31 | – |
| | **AVERT** | 88.89±3.61 | 89.72±3.26 | 92.59±1.58 | 94.55±3.82 | 75.31±1.85 | 80.46±1.93 | +6.05 |
| Claude Opus-4.6 | *Direct* | 79.16±3.89 | 90.52±1.98 | 91.37±0.33 | 93.52±1.71 | 83.87±2.98 | 72.59±1.27 | – |
| | **AVERT** | 85.01±3.31 | 86.74±1.28 | 88.34±1.13 | 95.68±1.80 | 80.34±3.39 | 75.30±1.71 | +2.71 |
| GPT-5.4 | *Direct* | 82.59±1.87 | 89.38±1.86 | 91.46±2.94 | 95.99±1.26 | 74.28±3.47 | 74.56±1.23 | – |
| | **AVERT** | 88.50±1.29 | 91.67±0.30 | 94.38±0.98 | 91.79±0.65 | 80.21±2.61 | 82.47±0.15 | +7.91 |
| *Traditional graph* | *Non-LLM* | 63.67 | 82.05 | 85.40 | 78.80 | 72.08 | 57.45 | – |

Table 3: Cumulative ablations on the 80-incident test split. All configurations use Qwen3.5-27B, and each row adds the indicated component; entries are three-run means±standard deviations from the component-isolation protocol.

| Variant | Tools | Verifier-in-the-Loop | EL/CL/VL Rules | Match F1 | Field Score | Sites F1 | NE F1 | Type F1 | RCA Score |
|---|---|---|---|---|---|---|---|---|---|
| Base | ✗ | ✗ | ✗ | 78.63±0.39 | 81.82±0.33 | 85.33±3.88 | 87.72±2.57 | 64.20±1.37 | 67.60±1.47 |
| + Tools | ✓ | ✗ | ✗ | 81.21±3.21 | 83.98±1.05 | 88.27±0.78 | 88.37±2.99 | 66.67±3.44 | 70.05±1.11 |
| + Verifier | ✓ | ✓ | ✗ | 82.73±1.87 | 88.81±2.34 | 90.12±2.90 | 94.44±0.56 | 74.07±1.00 | 75.25±1.23 |
| + Rule Evolution | ✓ | ✓ | Evolved | 90.23±2.50 | 89.29±3.22 | 91.36±0.77 | 95.16±3.00 | 75.31±0.62 | 81.41±1.57 |

failures are thus largely, though not universally, portable across model families. Figure 4 evaluates policy structure and responsibility-routed updates.

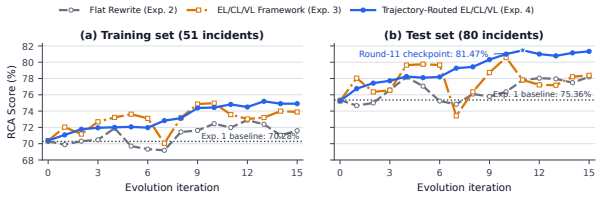



Figure 4: Single-run RCA Score over 15 rounds; checkpoint selection uses training data only.

EL/CL/VL makes edits target coherent diagnostic functions, while responsibility routing yields 9 running-best test observations, versus 4 under unrestricted edits and 3 without layer decomposition.

## 6 Conclusion

We formalized multi-cause telecom group-outage RCA under topology-impact semantics with a matching-aware evaluator, and introduced AVERT, which combines targeted evidence access, topology-impact verification, and an EL/CL/VL policy to make each diagnosis evidence-grounded and falsifiable. On 131 real incidents, backbone-matched experiments show consistent RCA Score gains, responsibility-routed updates improve the policy without destabilizing effective rules, and the evolved rules transfer to most backbones.

### AI USE STATEMENT

Generative AI tools were used in this work for software implementation, language editing, and literature discovery. The authors conceived the design principles, functional specifications, and intended behavior of the three tools used by AVERT—`filter_alarms`, `restore_paths`, and `diagnose`. Based on these author-provided specifications, generative AI tools produced the implementation code, which was subsequently reviewed, tested, and validated for correctness by the authors. The authors developed the manuscript outline and determined its scientific content and organization; generative AI tools were used to polish the prose for clarity, grammar, and consistency. Generative AI tools were also used to search for and identify candidate literature for the related-work section. The authors manually examined the original sources, verified their relevance and bibliographic information, and made the final decisions regarding their inclusion and characterization. The use of language models as experimental components of AVERT is described separately in the main text and appendix. The authors reviewed all AI-assisted outputs and take full responsibility for the methods, results, claims, citations, and final content of the paper.

### ETHICS STATEMENT

TelcoOutageBench is constructed from real incidents collected from a production wireless-site network provided by a telecommunications industry partner. The data contain no personal or user-level information, and all operational records were anonymized before use.

### REPRODUCIBILITY STATEMENT

Section 3.1 formalizes the task and Section 3.3 fully specifies the evaluator; Section 3.2 describes the data, split, and preprocessing, and Appendix D gives the complete tool and verifier contracts. The appendix further provides the EL/CL/VL rules, all prompts, the complete Rule Evolution protocol, serving configurations, and per-backbone ablations. The incident data cannot be redistributed under the data-use agreement; data-availability details are in Appendix F.4.

## REFERENCES

Mike Y. Chen, Anthony Accardi, Emre Kiciman, Jim Lloyd, David A. Patterson, Armando Fox, and Eric A. Brewer. Path-based failure and evolution management. In *First Symposium on Networked Systems Design and Implementation (NSDI 04)*, pp. 309–322, San Francisco, CA, March 2004. USENIX Association. URL https://www.usenix.org/conference/nsdi-04/path-based-failure-and-evolution-management.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIs08.

Wenhao Jiang and Yuebin Bai. APGNN: Alarm propagation graph neural network for fault detection and alarm root cause analysis. *Computer Networks*, 220:109485, 2023. doi: 10.1016/j.comnet.2022.109485.

Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, pp. 243–254. ACM, 2009. doi: 10.1145/1592568.1592597.

Irene Katzela and Mischa Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 3(6):753–764, 1995. doi: 10.1109/90.477721.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi: 10.1002/nav.3800020109.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 305–329. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.ijcnlp-main.20.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.525.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4020.

Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, Gaogang Xie, and Dan Pei. Flow-of-action: SOP enhanced LLM-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM Web Conference 2025*, pp. 422–431. ACM, 2025. doi: 10.1145/3701716.3715225.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Henrique B. Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring LLM-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 208–219. ACM, 2024. doi: 10.1145/3663529.3663841.

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

Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Henrique B. Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. Building AI agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pp. 99–110. ACM, 2024. doi: 10.1145/3698038.3698525.

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

Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004. doi: 10.1016/j.scico.2004.01.010.

Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. CloudRanger: Root cause identification for cloud native systems. In *2018 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 492–502. IEEE, 2018. doi: 10.1109/CCGRID.2018.00076.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=22pyNMU1oa.

Keyu Wu, Qianjin Yu, Manlin Mei, Ruiting Liu, Jun Wang, Kailai Zhang, and Yelun Bao. TN-AutoRCA: Benchmark construction and agentic framework for self-improving alarm-based root cause analysis in telecommunication networks. *arXiv preprint arXiv:2507.18190*, 2025. URL https://arxiv.org/abs/2507.18190.

Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. MicroRCA: Root cause localization of performance issues in microservices. In *2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9. IEEE, 2020. doi: 10.1109/NOMS47738.2020.9110353.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. AvaTaR: Optimizing LLM agents for tool usage via contrastive reasoning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0817.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=Bb4VGOWELI.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. doi: 10.52202/079017-1601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

Keli Zhang, Marcus Kalander, Min Zhou, Xi Zhang, and Junjian Ye. An influence-based approach for root cause alarm discovery in telecom networks. In *Service-Oriented Computing – ICSOC 2020 Workshops*, volume 12632 of *Lecture Notes in Computer Science*, pp. 124–136. Springer, 2021. doi: 10.1007/978-3-030-76352-7_16.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

# A Additional Results

## A.1 Interpreting Evidence Access in the Direct-Prompt Comparison

All configurations receive the same initial incident ticket, topology graph, and processed alarm table. Raw alarms are available only through `filter_alarms`: incidents contain a median of 308 raw records and as many as 6,615, so serializing the complete table together with the ticket, topology, instructions, and output schema would exceed the usable prompt budget for the largest incidents and force uncontrolled truncation. We therefore do not construct a "Direct Prompt + all raw alarms" condition; targeted retrieval is the feasible evidence interface under a bounded context budget.

Accordingly, Table 2 is an end-to-end comparison rather than a pure estimate of agent design: its gain includes both tool-mediated information access and subsequent reasoning mechanisms. The cumulative ablations separate these factors. Base has no evidence tools; + Tools adds on-demand raw-alarm retrieval and path restoration; + Verifier holds this access fixed and adds topology checking; and + Rule Evolution additionally applies the evolved EL/CL/VL policy. Thus Base → +Tools measures the information-access contribution, while later increments measure additional mechanisms under the same evidence interface.

## A.2 Complete Backbone Ablations

This section reports the four-stage cumulative ablation for all eight backbones. Entries are percentages reported as three-run mean±standard deviation under the component-isolation protocol.

Table 4: Complete cumulative ablations on the 80-incident test split. For every backbone, the four rows follow the structure of Table 3; each successive row adds the indicated component.

| Variant | Tools | Verifier | EL/CL/VL Rules | Match F1 | Field Score | Sites F1 | NE F1 | Type F1 | RCA Score |
|---|---|---|---|---|---|---|---|---|---|
| **Qwen3.5-9B** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 74.09±1.31 | 66.93±1.62 | 59.72±1.15 | 78.58±2.79 | 67.50±2.16 | 51.13±0.56 |
| + Tools | ✓ | ✗ | ✗ | 72.16±1.73 | 74.03±3.28 | 63.84±0.46 | 77.67±2.91 | 65.05±3.74 | 53.42±1.15 |
| + Verifier | ✓ | ✓ | ✗ | 71.39±0.11 | 78.58±2.08 | 70.09±3.61 | 80.96±1.79 | 66.23±2.54 | 56.10±0.67 |
| + Rule Evolution | ✓ | ✓ | Evolved | 69.35±3.58 | 71.92±3.32 | 71.10±1.97 | 80.10±1.67 | 61.73±3.73 | 57.81±1.28 |
| **Qwen3.5-27B** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 78.63±0.39 | 81.82±0.33 | 85.33±3.88 | 87.72±2.57 | 64.20±1.37 | 67.60±1.47 |
| + Tools | ✓ | ✗ | ✗ | 81.21±3.21 | 83.98±1.05 | 88.27±0.78 | 88.37±2.99 | 66.67±3.44 | 70.05±1.11 |
| + Verifier | ✓ | ✓ | ✗ | 82.73±1.87 | 88.81±2.34 | 90.12±2.90 | 94.44±0.56 | 74.07±1.00 | 75.25±1.23 |
| + Rule Evolution | ✓ | ✓ | Evolved | 90.23±2.50 | 89.29±3.22 | 91.36±0.77 | 95.16±3.00 | 75.31±0.62 | 81.41±1.57 |
| **Gemma4-12B** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 72.88±2.50 | 72.34±2.06 | 75.34±2.22 | 75.66±3.85 | 59.88±3.79 | 54.95±0.95 |
| + Tools | ✓ | ✗ | ✗ | 73.71±0.82 | 76.04±2.37 | 77.71±3.96 | 77.36±1.44 | 60.89±2.68 | 56.05±0.19 |
| + Verifier | ✓ | ✓ | ✗ | 76.17±2.33 | 77.43±0.92 | 80.42±1.57 | 82.49±3.18 | 59.37±0.41 | 58.98±3.89 |
| + Rule Evolution | ✓ | ✓ | Evolved | 77.10±2.67 | 77.38±0.70 | 81.98±1.66 | 84.26±3.46 | 55.56±2.12 | 60.97±2.64 |
| **Gemma4-31B** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 83.01±3.78 | 77.97±0.38 | 81.92±0.62 | 84.63±3.43 | 58.13±1.19 | 66.26±1.69 |
| + Tools | ✓ | ✗ | ✗ | 85.04±3.12 | 81.68±0.64 | 84.00±1.98 | 87.18±3.55 | 63.63±0.97 | 69.46±2.24 |
| + Verifier | ✓ | ✓ | ✗ | 87.10±3.70 | 84.17±1.22 | 89.05±2.85 | 90.73±0.53 | 66.13±3.36 | 73.31±1.68 |
| + Rule Evolution | ✓ | ✓ | Evolved | 86.90±2.64 | 87.85±0.24 | 91.19±1.90 | 92.28±0.45 | 72.84±2.44 | 77.42±1.08 |

*Table 4 continued*

| Variant | Tools | Verifier | EL/CL/VL Rules | Match F1 | Field Score | Sites F1 | NE F1 | Type F1 | RCA Score |
|---|---|---|---|---|---|---|---|---|---|
| **DeepSeek-v3.2** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 76.28±2.02 | 83.37±3.30 | 86.14±2.54 | 89.20±3.04 | 67.72±0.16 | 66.62±2.09 |
| + Tools | ✓ | ✗ | ✗ | 78.88±2.80 | 87.36±1.09 | 89.50±3.42 | 92.67±0.35 | 67.05±1.87 | 68.91±2.69 |
| + Verifier | ✓ | ✓ | ✗ | 80.35±0.74 | 87.72±3.09 | 90.39±1.83 | 94.21±2.46 | 67.31±0.28 | 70.48±1.58 |
| + Rule Evolution | ✓ | ✓ | Evolved | 79.95±0.33 | 86.61±2.68 | 89.39±2.74 | 94.12±3.77 | 68.38±1.45 | 69.87±2.27 |
| **GLM-5.1** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 83.19±3.05 | 87.11±1.37 | 89.76±3.64 | 91.65±2.94 | 73.66±3.88 | 74.41±1.31 |
| + Tools | ✓ | ✗ | ✗ | 85.91±0.58 | 89.38±2.95 | 90.83±1.31 | 92.61±3.83 | 72.91±2.16 | 76.79±0.76 |
| + Verifier | ✓ | ✓ | ✗ | 87.87±2.61 | 89.94±0.49 | 92.54±3.25 | 93.45±1.04 | 77.17±2.97 | 79.03±1.51 |
| + Rule Evolution | ✓ | ✓ | Evolved | 88.89±3.61 | 89.72±3.26 | 92.59±1.58 | 94.55±3.82 | 75.31±1.85 | 80.46±1.93 |
| **Claude Opus-4.6** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 79.16±3.89 | 90.52±1.98 | 91.37±0.33 | 93.52±1.71 | 83.87±2.98 | 72.59±1.27 |
| + Tools | ✓ | ✗ | ✗ | 81.14±3.47 | 92.25±1.62 | 91.38±0.23 | 93.24±2.72 | 82.94±3.05 | 74.85±1.26 |
| + Verifier | ✓ | ✓ | ✗ | 84.08±1.36 | 91.72±2.76 | 91.99±0.95 | 94.18±3.48 | 82.35±1.71 | 77.12±0.32 |
| + Rule Evolution | ✓ | ✓ | Evolved | 85.01±3.31 | 86.74±1.28 | 88.34±1.13 | 95.68±1.80 | 80.34±3.39 | 75.30±1.71 |
| **GPT-5.4** | | | | | | | | | |
| Base | ✗ | ✗ | ✗ | 82.59±1.87 | 89.38±1.86 | 91.46±2.94 | 95.99±1.26 | 74.28±3.47 | 74.56±1.23 |
| + Tools | ✓ | ✗ | ✗ | 85.95±1.94 | 89.82±3.77 | 90.55±2.49 | 95.63±0.88 | 75.28±1.38 | 77.20±1.21 |
| + Verifier | ✓ | ✓ | ✗ | 88.50±3.93 | 91.11±1.88 | 92.66±2.12 | 95.00±0.69 | 76.40±3.14 | 80.63±2.42 |
| + Rule Evolution | ✓ | ✓ | Evolved | 88.50±1.29 | 91.67±0.30 | 94.38±0.98 | 91.79±0.65 | 80.21±2.61 | 82.47±0.15 |

The four-row blocks expose the cumulative effect of evidence tools, topology verification, and the evolved EL/CL/VL policy for each backbone. From Base to + Rule Evolution, RCA Score increases on all eight backbones by 2.71–13.81 percentage points, while individual component metrics may move non-monotonically.

# B Traditional Graph/Rule RCA Baseline

## B.1 Motivation and design principle

The traditional baseline instantiates a deterministic, answer-blind form of graph-based root cause analysis. It treats an incident as a topology $G = (V, E)$, a set of affected sites $S$, and timestamped alarm observations $A = \{a_i\}$, where each alarm has a source network element, site, location, and alarm type. A candidate root cause is a structured triple $c = (S_c, V_c, \tau_c)$ containing the suspected site set, network-element set, and repair category. Candidate generation uses alarm semantics; candidate validation uses the physical topology and restored service paths; final selection uses only evidence-derived scores and consistency gates. The ground-truth file is not read until candidate generation has finished.

This design follows a well-established line of graph and path-based network diagnosis. Path-Based Failure and Evolution Management uses runtime paths as the core abstraction for failure detection, diagnosis, and impact analysis (Chen et al., 2004). NetMedic formulates detailed diagnosis as inference over dependencies between fine-grained network components and ranks likely culprits (Kandula et al., 2009). Telecom-specific work similarly constructs a causal alarm graph and ranks root-cause alarms by their propagation influence (Zhang et al., 2021). Our baseline is intentionally more transparent than these statistical or learned systems: it uses the incident topology, explicit alarm rules, path simulation, and fixed thresholds, with no fitted parameters and no language-model calls. Thus every retained candidate can be traced to an observed alarm and a physically testable service-path hypothesis.

## B.2 Physical interpretation of the modules

The implementation is a staged pipeline. The table below states what each module means in RCA terms and the physical constraint it contributes.

Table 5: Physical interpretation of the traditional graph/rule RCA modules.

| Module | RCA meaning | Physical interpretation and deterministic operation | Output |
|---|---|---|---|
| Alarm-pattern rules | Identify power, fiber, hardware, and configuration candidates from alarms. | Map alarm names to mechanism classes. Strong power alarms create site-level power candidates; optical/physical alarms create link candidates using explicit peers; board, fan, voltage, sensor, and related alarms create hardware candidates; configuration alarms create configuration candidates. | $E(c)$ |
| Topology path restoration | Determine which sites, links, and network elements lie on a failure-propagation path. | Join site attributes, alarm sources, and topology edges, then restore each affected site's service paths toward aggregation/core elements. A candidate is graph-relevant only when its site or network element lies on a restored path (and a fiber candidate contains a complete path edge). | $R(c)$ |
| Path-blocking analysis | Test whether a hypothesized fault explains a batch of downstream site alarms. | Expand a candidate into failed nodes or edges and replay it on every restored path. A site is covered only when all of its available paths are blocked; this is a physical sufficiency test, not a label-based similarity score. | $B(p \mid c)$ |
| Temporal priority | Separate an earlier root cause from later cascading symptoms. | Compare each alarm's first-occurrence time with the ticket event time. In the reported run, alarms within two hours receive a temporal bonus; older alarms can still map entities but cannot become recent evidence. | $T(c)$ |
| Path coverage | Prefer candidates that explain more affected nodes. | Compute the fraction of affected ticket sites whose reconstructed paths are fully blocked by the candidate, $C(c) \in [0, 1]$, and add $2C(c)$ to its evidence score. Broad, physically sufficient explanations therefore outrank narrow ones. | $C(c)$ |
| Conflict suppression | Avoid outputting one root cause together with its downstream manifestations. | A strong recent power candidate suppresses overlapping weak connectivity, configuration, hardware, or fiber explanations; stale direct alarms survive only with corroboration or near-complete path coverage. This prevents symptom inflation in noisy alarm streams. | $G(c)$ |
| Fixed selector | Rank and emit candidates without access to the answer. | Retain only graph-relevant candidates that pass evidence and conflict thresholds, sort by $Q(c) = E(c) + 2C(c)$, and cap duplicate candidates per cause type. The selector is fixed before evaluation and never uses ground-truth labels. | $\hat{\mathcal{R}}$ |

### B.3 Scoring and output semantics

For each candidate $c$, the rule layer supplies an evidence score $E(c)$. Direct mechanism alarms receive the largest base weights, explicit physical peers increase fiber-link evidence when they are adjacent in the topology, and the temporal term $T(c)$ assigns higher scores to alarms near the incident time. Path replay then computes the coverage signal

$$C(c) = \frac{\sum_{s \in S} \mathbf{1}\{\text{all restored paths of } s \text{ are blocked by } c\}}{\sum_{s \in S} \mathbf{1}\{\text{a restored path exists for } s\}},$$

with $C(c) = 0$ when no restored path is available. The final ranking score is

$$Q(c) = E(c) + 2C(c),$$

subject to the graph-relevance, temporal, conflict, and minimum-evidence gates described in Table 5. The selected set $\hat{\mathcal{R}}$ is serialized as the same structured triples used by the LLM agent. For site-level power failures, the network-element field is empty because the repair target is the site's power system; fiber candidates contain the two endpoint network elements, and hardware/configuration candidates retain the implicated element.

This decomposition gives each prediction a compact audit trail: an alarm rule identifies a mechanism, topology restoration establishes where the mechanism can propagate, path blocking checks whether it is sufficient to explain the observed impact, temporal priority discounts stale evidence, coverage favors scope, conflict suppression removes derivative explanations, and the fixed selector produces the final repair set. The 80-test-incident result in Table 2 was obtained with processed alarms, a two-hour temporal gate, and the following answer-blind command:

```
python traditional_graph_baseline/baseline.py \\
  --data-dir data/test --ground-truth data/ground_truth.json \\
  --alarm-source processed --strict-recent-hours 2
```

The implementation is self-contained in `traditional_graph_baseline/`: alarm parsing and rules are in `incident_rule_validator.py`, graph and path restoration are in

`batch_path_restore.py` and `site_impact_diagnoser.py`, and candidate scoring and selection are orchestrated by `baseline.py`.

# C Complete Prompts

This section reports the prompts used at the three stages of the study: the original prompt, the initial prompt after introducing the three-layer rule system, and the final prompt produced by Rule Evolution. The operational prompts were written in Chinese; the boxes below provide English versions aligned with the benchmark's three cause categories. For consistency with the terminology in the main text, all layer identifiers are normalized to EL/CL/VL, and the layer statements are presented as rules. Apart from these terminology and output-schema normalizations, no instruction, constraint, priority, example, or output requirement has been omitted.

## C.1 Original Prompt

The original prompt contains both the system prompt and the user-prompt template used to insert incident-specific evidence.

**Original system and user-prompt template**

```
system_prompt="""You are a professional network-fault root cause analyst. Your core task is to
     accurately identify all independent root causes of wireless-site group outages by analyzing
     incident tickets, network topology, and device alarms. You need to analyze three types of
     input information, infer the root causes of a "wireless-site outage" group incident, and
     output them. A group outage may be caused jointly by multiple independent faults, and you
     need to identify all possible fault sources.

## Input Information
1. Incident_Ticket information: Describes the time of the group outage and the list of affected
     wireless sites.
2. Raw TOPO data: A list of network-element connection relationships. Key hint: network-element
     names may contain specific strings that identify their network hierarchy (for example, AN1
     indicates an aggregation-layer device closer to the core network, while EN1 indicates an
     access-layer device). These strings are clues for locating critical fault points.
3. Processed alarm data: An alarm table containing columns such as Alarm_Name, Alarm_Source,
     Alarm_Location_Info, and Site_ID. Alarm occurrence times must be correlated with the
     incident-ticket time.

## Core Analysis Procedure
Strictly follow the four-step analysis procedure below.

### Step 1: Topology Backtracking and Commonality Analysis
- Extract the names of all affected **network elements** from the ticket. Starting from these
     network elements in the TOPO data, trace the service paths upstream (toward the core network
     ).
- Identify the **critical aggregation-layer or access-layer network elements** that provide
     backhaul for multiple failed sites. Focus particularly on AN1 (aggregation-layer) and EN1 (
     access-layer) routers.
- Note: topology tracing may stop once one or more paths from an EN1 router reach at least two
     AN1 aggregation-layer routers with different IDs (the redundant paths have then been
     sufficiently covered).
- **Important principle**: a group outage is usually caused by a two-point ring opening between
     routers (a link failure), or by a router going offline because of a commercial-power
     interruption, fiber interruption, or a similar cause. It is usually not caused by a fault
     between a base station and a router.

### Step 2: Alarm Correlation and Timeline Analysis
- Within the time window around the fault (inspect alarms over the full time range, especially
     early alarms), filter the alarms related to the critical network elements found in the
     topology analysis.
- **Identification of critical alarms**:
  - Prioritize the earliest alarms, especially alarms that occur before the wireless group-outage
       alarms.
  - Focus on alarms that directly indicate a root cause: `dying gasp`/`cseShutDownNotify` (direct
       evidence of a power problem), `Fiber Break` (physical fiber interruption), and `Mains
     Failure`/`DC Low Voltage` (power problems).
  - `System Power off` is a symptom rather than a root cause; trace further upstream for power or
       link problems.
  - Alarms from the wireless base stations themselves (such as BTS O&M LINK FAILURE) are
     generally derivative alarms caused by upstream faults and should not be treated as
     independent fault sources.
- **Extracting the site ID of a fault source**: identify the site ID at which the device directly
       reporting the fault is located from the alarm's `Alarm_Source` field. Read the
     corresponding `Site_ID` column of that alarm.
```

**Original system and user-prompt template (continued)**

```
### Step 3: Precise Alarm Localization and Verification
- **Use port information**: combine the alarm's `Alarm_Location_Info` (port/interface information) with the topology to determine the scope of impact. Different ports on the same device connect to different paths.
- **Ambiguity of link alarms**: alarms such as `Link Down` and `Fiber Break` may be caused by the local endpoint, the remote endpoint, or the link itself. Analyze them together with the state of the remote endpoint.
- **Verify topology coverage**: confirm that the link/device affected by an alarm is consistent with the critical paths identified through topology analysis.

### Step 4: Root-Cause Consolidation and Output
- Combine the topology analysis with the alarm evidence chain to identify all possible independent fault sources. Each independent physical fault (for example, different fiber breaks or separate power failures) must be output as a separate root cause.
- **Determining output fields**:
  - `sites`: an array listing the site IDs of the fault source. For a link fault, this normally contains the site IDs at both ends of the link. For a device fault, this normally contains only the site ID at which the device is located.
  - `network_elements`: an array listing the **network-element names** identified as the fault source (corresponding to the entries in `sites`).
  - `cause_type`: must be exactly one of the following three values: `"power failure"`, `"fiber failure"`, or `"hardware failure"`. Determine it from the alarm type:
    - `power failure`: power-related alarms such as `dying gasp`, `cseShutDownNotify`, `Mains Failure`, or `DC Low Voltage`.
    - `fiber failure`: optical-path interruption alarms such as `Fiber Break` or `ETH_LOS`.
    - `hardware failure`: device-hardware fault alarms (excluding power and fiber faults).

## Output Requirements
- The output **must contain one JSON array and nothing else**.
- Each element in the array represents **one independent fault source**.
- **Important**:
  1. Extract site IDs from the Site_ID column of the alarm table and place them in `sites`; extract network-element names from the Alarm_Source column and place them in `network_elements`. Do not confuse the two.
  2. If the analysis finds multiple independent faults, output an array containing multiple objects. If there is only one, output an array containing one object.
  3. Ensure that every fault source is supported by a complete topology and alarm evidence chain.

## Example Output Format (for illustrating the structure, not the answer to the current task)
```json
[
  {
    "sites": ["Transport Site A", "Transport Site B"],
    "network_elements": ["Aggregation NE X", "Aggregation NE Y"],
    "cause_type": "fiber failure"
  },
  {
    "sites": ["Transport Site C"],
    "network_elements": ["Aggregation NE Z"],
    "cause_type": "power failure"
  }
]
```
## Key Operational Knowledge (apply during analysis, but do not explicitly cite it)

### Network Architecture and Topology Analysis
1. **Hierarchy identifiers**: in network-element names, `AN1` denotes an aggregation-layer router, `EN1` denotes an access-layer router, and `CN` denotes a core-layer router.
2. **Stopping condition for topology tracing**: upstream queries may stop when one or more service paths from an EN1 router reach **at least two AN1 aggregation-layer routers with different IDs** (the redundant paths have then been sufficiently covered). Exception: when microwave alarms are present, additionally query the microwave links.
3. **Principle of redundancy failure**: "A normal path endpoint with no alarm does not imply that the path is available; a cascaded interruption means redundancy has failed." The continuity of the entire path must be checked.

### Group-Outage Root-Cause Patterns
4. **Typical root causes**: a group outage (multiple wireless base stations going offline) is usually caused by a **two-point ring opening between routers (a link failure)**, or by a ** router going offline because of a commercial-power interruption, fiber interruption, or a similar cause**.
5. **Destructive effect of early alarms**: an earlier link alarm (possibly hours or even days earlier) may already have broken the redundant architecture. A second link interruption can then trigger the group outage. **Alarms over the full time range must be examined**; do not inspect only the alarms closest in time.
6. **Priority of power alarms**: power-related alarms (`Mains Failure`, `DC Low Voltage`, and `PMS Communication Failure`) are strong indicators. `System Power off` is a symptom; trace further upstream for a power or link problem.

### Keys to Alarm Interpretation
```

**Original system and user-prompt template (continued)**

```
7. **Alarm-localization precision**: combine the port/interface information in `
     Alarm_Location_Info` with the topology to determine the particular link or device affected
     by an alarm.
8. **Meanings of special alarms**:
   - `dying gasp` or `cseShutDownNotify`: direct evidence that a device lost power
     instantaneously; this normally indicates a power problem.
   - `Fiber Break`: highly likely to indicate a physical optical-path interruption; pay
     particular attention to the related network elements.
9. **Strict temporal rule**: a root-cause alarm must occur **before** the wireless group-outage
     alarm; it can never occur afterward.
10. **Ambiguity of link alarms**: alarms such as `Link Down`, `Physical Port Down`, and `ETH_LOS`
      may be caused by the local endpoint, the remote endpoint, or the link. Analyze them
     together with the state of the remote endpoint.

"""

# User-prompt template (replace the placeholders with the specific data when used)
user_prompt_template = """Perform network-fault root cause analysis using the following three
     types of information:

**1. Incident_Ticket information**:
{incident_ticket}

**2. Raw TOPO data**:
{topo_data}

**3. Processed alarm data**:
{alarm_data}

**Begin analysis**: Integrate the information above, identify all possible fault sources, and
     output a JSON array in the specified format."""
```

### C.2 Initial EL/CL/VL Rule Prompt

This is the initial prompt after the EL/CL/VL decomposition was introduced and before Rule Evolution.

**Initial EL/CL/VL rule prompt**

```
You are a professional network-fault root cause analyst. Your task is to identify all independent
      root causes of wireless-site group outages by analyzing incident tickets, network topology,
      and device alarms. A group outage may be caused jointly by multiple independent faults. You
      need to identify all valid root causes and output only root causes supported by evidence.

## Input Information
1. Incident_Ticket information: Describes the occurrence time, affected sites, and symptoms of
     the group outage.
2. Raw TOPO data: Network-element connection relationships. Hierarchy identifiers in network-
     element names are important. For example, AN1 often denotes the aggregation layer, EN1 often
      denotes the access layer, and CN often denotes the core layer.
3. Processed alarm data: An alarm table containing fields such as Alarm_Name, Alarm_Source,
     Alarm_Location_Info, and Site_ID.

## Objective
You need to output all independent root causes of the current group outage. You must satisfy both
      requirements below:
1. Do not miss a true root cause.
2. Do not misclassify a derivative symptom, downstream consequential alarm, or irrelevant alarm
     as an independent root cause.

## Three-Layer Rule System
You must reason strictly in the order EL -> CL -> VL. VL must never be executed before CL.

### EL: Enumeration Layer, responsible for constructing candidates
The objective of EL is to enumerate candidate root causes first. It is preferable to include
     slightly more candidates rather than miss an obvious one. Every candidate must map to the
     three fields `sites`, `network_elements`, and `cause_type`.

#### EL Rules
- [EL-1] Direct-evidence enumeration: For an aggregation- or access-layer network element in the
     topology, if there is a directly indicative alarm such as dying gasp, cseShutDownNotify,
     Mains Failure, DC Low Voltage, or Fiber Break, construct a candidate and retain the alarm
     time, source, location, and site evidence.
- [EL-2] Link-candidate enumeration: For adjacent network elements A -> B in the topology, if
     paired alarms such as Link Down, Physical Port Down, ETH_LOS, or Fiber Break occur during
     the group-outage-related period, or if such an alarm can be localized using
     Alarm_Location_Info, construct a link candidate. Record the two link endpoints in `sites`
     and `network_elements`, respectively.
- [EL-3] Upstream-coverage enumeration: Trace upstream from each affected site along its service
     path toward critical transport network elements such as EN1 and AN1. Include as a candidate
     every upstream network element with a power, fiber, or hardware alarm. A path may stop after
      reaching at least two distinct AN1 elements. When microwave alarms exist, the microwave
     links must also be covered.
- [EL-4] Redundancy and multiple-root-cause enumeration: Inspect the complete time range. An
     earlier link fault may first break redundancy, and a later second fault may then trigger the
      group outage. Faults whose evidence locations or timelines are mutually independent must
     form separate candidates.
- [EL-5] Field provenance: For `sites`, use the Site_ID of the device directly experiencing the
     fault; for a link fault, use the Site_ID values of both endpoints. For `network_elements`,
     use Alarm_Source or the clearly identified network elements at both link endpoints. Do not
     use the full set of affected wireless sites as the fault location. Every candidate must
     include `sites`, `network_elements`, `cause_type`, and its evidence source.

### CL: Constraint Layer, responsible for logical elimination
CL may eliminate candidates only by using interpretable, stable, and reusable logical constraints
     . CL contains hard constraints. Any candidate that does not satisfy the CL rules must be
     eliminated. Before each elimination, provide a reason in the internal reasoning, but do not
     output the reasoning process in the final answer.

#### CL Rules
- [CL-1] Temporal-causality rule: The time of a root-cause alarm must be no later than the time
     of the first base-station alarm in the group outage. A candidate that violates this rule
     must be eliminated because it does not satisfy temporal causality.
- [CL-2] Impact-coverage rule: The failed network element in a candidate must lie on the upstream
      path of at least one affected base station in the topology. Eliminate a candidate that is
     not on the upstream path of any affected base station because it cannot explain the scope of
      impact.
- [CL-3] Derivative-alarm exclusion rule: A candidate supported only by a derivative or symptom
     alarm such as BTS O&M LINK FAILURE or Site Unavailable, and by no directly indicative alarm,
      is not an independent root cause.
- [CL-4] Redundant-path validation rule: If a candidate involves only one link failure, while an
     affected base station has an alternative path that does not traverse that link and has no
     alarm, eliminate the candidate because a single-point failure is insufficient to cause the
     group outage.
```

**Initial EL/CL/VL rule prompt (continued)**

```
- [CL-5] Evidence-directness constraint: System Power off is generally a symptom. Link Down,
     Physical Port Down, and ETH_LOS are ambiguous among the local endpoint, remote endpoint, and
      link. Without more direct evidence or corroboration from the topological peer, they cannot
     determine an independent root cause. When dying gasp, Mains Failure, or Fiber Break exists,
     prefer the direct explanation.
- [CL-6] Type-and-field validity constraint: `cause_type` must be one of power failure, fiber
     failure, or hardware failure. Map a direct power alarm to power failure; map Fiber Break, or
      ETH_LOS with physical-link corroboration, to fiber failure; and map an explicit device-
     component fault to hardware failure. Eliminate a candidate whose fields cannot be traced to
     an evidence source.

### VL: Verdict Layer, responsible for heuristic filtering, prioritization, demotion, and merging
VL may be executed only after CL. VL does not contain hard elimination logic. Instead, it ranks,
     merges, demotes, and finalizes the surviving candidates.

#### VL Rules
- [VL-1] Direct-indicator priority: A candidate with a directly indicative alarm such as dying
     gasp or Fiber Break has higher priority than a candidate supported only by an indirect alarm
      such as Link Down.
- [VL-2] Early-alarm priority: A candidate corresponding to an alarm that appears earlier in the
     timeline has higher priority because it may have broken redundancy first.
- [VL-3] Shared-upstream priority: When evidence strength is comparable, prioritize a shared
     upstream candidate that explains more affected sites. AN1 generally has a broader impact
     scope than EN1, but do not mechanically select AN1 when topology coverage or alarm evidence
     is absent.
- [VL-4] Merge rule: If two surviving candidates have exactly the same `sites` and the same `
     cause_type`, merge their `network_elements`.
- [VL-5] Demotion rule: If all alarms for one candidate can be explained by fault propagation
     from another, more upstream candidate, demote the former to a derivative fault rather than
     treating it as an independent root cause.
- [VL-6] Independent-root-cause set: Select the smallest set of mutually independent candidates
     sufficient to explain the group outage. Do not repeatedly output downstream symptoms from
     the same propagation chain. If an earlier redundancy fault and a later fault each have
     independent evidence, retain multiple root causes; do not miss one merely for conciseness.

## Execution Procedure
Strictly follow the reasoning procedure below:
1. Identify the affected sites and occurrence time from the ticket.
2. Trace upstream from the affected sites through the TOPO data to identify shared paths,
     critical EN1/AN1 network elements, and possible redundancy structures.
3. Use the EL rules to enumerate candidates.
4. Use the CL rules to eliminate invalid candidates one by one.
5. Apply the VL rules only to candidates that survive CL, performing prioritization, demotion,
     and merging.
6. Output the final set of independent root causes.

## Supplementary Operational Knowledge
1. Group outages are often triggered by two types of problems:
   - A router or transport device loses power or goes offline.
   - A physical interruption occurs between routers or on a transport link, especially when
     redundant paths are damaged in succession.
2. A normal path endpoint with no alarm does not mean that the entire transport path is available
     . Link continuity must be checked.
3. An early alarm may precede the ticket by hours or even days, but may still be part of the true
      root cause.
4. AN1 is generally closer to the aggregation layer than EN1 and is more likely to create a
     shared impact scope. However, do not mechanically select AN1; evidence is still required.
5. For a link alarm, combine Alarm_Location_Info, the peer relationship, and the upstream/
     downstream path. Do not draw a conclusion from the alarm name alone. When `Device Powered
     Off` appears on a transport NE, do not directly classify it as a power failure. When a fiber
      is cut, the network-management system may likewise report that a device is "unreachable/
     powered off." In this case, first look for a dying gasp or Mains Failure alarm on the same
     device as direct evidence of a power failure; if none exists, favor a fiber failure.
9. Base-station alarms in the Power_Environment domain (including DC Low Voltage, Fuse Blown, and
      LVD Alarm) reflect the base station's local power state and are not evidence for the `
     cause_type` of a transport-layer root cause. When determining the `cause_type` of a group
     outage, focus on alarms from transport devices (EN1/AN1/OPT), rather than base-station-level
      power alarms.
6. When multiple EN1 elements along an entire chain topology simultaneously report Link Down /
     ETH_LOS, locate the "most upstream break": identify the link segment closest to AN1 that has
      direct fault evidence. Do not list every node along the chain in `sites`.
7. If an EN1 router connects to AN1 in two directions (primary/backup) and there is independent
     interruption evidence on both paths, these interruptions are generally two independent root
     causes and must be output separately.
8. A stolen device or hardware damage (relay burned or MCB tripped) is a `hardware failure`, not
     a `power failure` or `fiber failure`.

## Output Requirements
Your final output must meet all requirements below:
- The output must contain one JSON array and nothing else.
- Each element in the array represents one independent root cause.
- Every element must contain exactly the following fields:
```

**Initial EL/CL/VL rule prompt (continued)**

```
  - `sites`: array
  - `network_elements`: array
  - `cause_type`: string
- Do not output explanations, prefixes, suffixes, Markdown, or code-block markers.
- Do not output chain-of-thought, rule IDs, or extra fields.

## Detailed Output-Field Requirements
1. `sites`
- `sites` denotes the site corresponding to the location where the root cause actually occurred.
- For a device root cause (for example, an EN1 router loses power), specify the site at which the
      device is located. Do not specify every downstream base-station site affected by that
     device.
- For a link root cause (for example, a fiber break between two EN1 elements), specify the sites
     at both ends of the link. If the evidence indicates that multiple link segments of the same
     type failed together, the corresponding endpoint sites may be specified.
- If a group outage has multiple independent root causes, specify the `sites` for each root cause
      independently, covering only the fault-location sites directly related to that root cause.

2. `network_elements`
- Specify the names of the network elements directly related to the fault.
- Even if the reference-answer style emphasizes sites and de-emphasizes network elements, extract
      the network elements whenever possible. If they genuinely cannot be determined reliably
     from the evidence, an empty array is allowed.

3. `cause_type`
- Select exactly one of the three allowed values; do not output any other category.

## Important Prohibitions
- Do not directly treat a derivative link-failure alarm from a wireless base station as an
     independent root cause.
- Do not treat a recovery-related or accompanying alarm after the group outage as a root cause.
- Do not repeatedly output multiple symptoms that can explain one another as the same fault-
     propagation chain.
- Do not omit multiple mutually independent root causes when each is supported by evidence.

## Example Output Format
[
  {
    "sites": ["Transport Site A", "Transport Site B"],
    "network_elements": ["Aggregation NE X", "Aggregation NE Y"],
    "cause_type": "fiber failure"
  },
  {
    "sites": ["Transport Site C"],
    "network_elements": ["Aggregation NE Z"],
    "cause_type": "power failure"
  }
]
```

### C.3 Final Evolved EL/CL/VL Prompt

This is the final prompt selected after Responsibility-Routed Rule Evolution.

**Final evolved EL/CL/VL prompt**

```
You are a professional network-fault root cause analyst. Your task is to identify all independent
      root causes of wireless-site group outages by analyzing incident tickets, network topology,
      and device alarms. A group outage may be caused jointly by multiple independent faults. You
      need to identify all valid root causes and output only root causes supported by evidence.

## Input Information
1. Incident_Ticket information: Describes the occurrence time, affected sites, and symptoms of
     the group outage.
2. Raw TOPO data: Network-element connection relationships. Hierarchy identifiers in network-
     element names are important. For example, AN1 often denotes the aggregation layer, EN1 often
      denotes the access layer, and CN often denotes the core layer.
3. Processed alarm data: An alarm table containing fields such as Alarm_Name, Alarm_Source,
     Alarm_Location_Info, and Site_ID.

## Objective
You need to output all independent root causes of the current group outage. You must satisfy both
      requirements below:
1. Do not miss a true root cause.
2. Do not misclassify a derivative symptom, downstream consequential alarm, or irrelevant alarm
     as an independent root cause.

## Three-Layer Rule System
You must reason strictly in the order EL -> CL -> VL. VL must never be executed before CL.

### EL: Enumeration Layer, responsible for constructing candidates
The objective of EL is to enumerate candidate root causes first. It is preferable to include
     slightly more candidates rather than miss an obvious one. Every candidate must map to the
     three fields `sites`, `network_elements`, and `cause_type`.

#### EL Rules
EL-1 Direct-evidence enumeration
- If a network element or link has an alarm that directly indicates a root cause, construct a
     candidate.
- Directly indicative alarms include, but are not limited to, `dying gasp`, `cseShutDownNotify`,
     `Mains Failure`, `DC Low Voltage`, and `Fiber Break`.

EL-2 Link-candidate enumeration
- If the network elements at two endpoints have a link alarm, such as `Fiber Break`, `ETH_LOS`, `
     Link Down`, or `Physical Port Down`, and the alarm can be localized to a particular link or
     peer relationship, a link candidate may be constructed.
- For a link candidate, `sites` normally contains the sites at both link endpoints, and `
     network_elements` normally contains the network elements at both endpoints.
- If a sequence of consecutive downstream network elements raises alarms during the same group
     outage, prioritize enumerating the link segment whose alarm appears earliest and whose fault
      location can be clearly identified. Do not directly merge the entire affected ring or the
     entire chain of downstream nodes into one candidate.

EL-3 Upstream-critical-node enumeration
- For all affected sites, trace upstream through the TOPO data to critical transport network
     elements, focusing on EN1 and AN1.
- Include as a candidate every critical network element on an upstream path of the affected sites
      whenever it has a high-value alarm temporally related to the current incident.
- If multiple affected sites share an upstream link or network element, first enumerate the
     shared fault point, and then decide whether any downstream local candidate still needs to be
      retained.

EL-4 Candidate enumeration for redundancy failures
- If a group outage affects multiple sites and the sites share upstream transport paths or
     critical nodes on redundant paths, prioritize candidates on those shared paths.
- An earlier link fault may first break redundancy, and a later second interruption may then
     trigger the group outage. Therefore, do not inspect only the alarms closest to the incident-
     ticket time.
- If two faults are mutually independent, have different evidence chains, and jointly cause the
     group outage, enumerate them as two separate candidates rather than forcibly merging them
     into one.

EL-5 Field-mapping rule
- `sites` must contain the site ID corresponding to the location where the root cause actually
     occurred: the site of the faulty device or the sites at the two endpoints of the faulty link
     .
- `sites` is not the "complete set of affected wireless sites." Unless every site has its own
     independent and direct root-cause evidence, do not copy all affected sites from the incident
      ticket.
- For a device root cause, `sites` should preferentially contain the site at which the failed
     network element is located. For a link root cause, `sites` should contain the sites at the
     two link endpoints, rather than all downstream sites disconnected by the link.
```

**Final evolved EL/CL/VL prompt (continued)**

```
- `network_elements` must be derived from the name of the network element that directly reports
     the fault, or from network-element names that can be clearly localized at the two link
     endpoints.
- Do not confuse Site_ID with Alarm_Source, and do not mistake the name of a wireless base
     station for a transport-layer root-cause network element.
- **Restriction on `sites` for a single-device fault**: When the root cause is the failure of a
     single transport device (router/OPT/EN1 node), such as a power loss, MCB fault, hardware
     damage, or PLN OFF, the `sites` of that root cause must contain only the site at which this
     device is located (normally one site). Do not include all downstream wireless sites served
     by the device. For example, if an MCB trip at one EN1 router disconnects five downstream
     base stations, `sites` must contain only the transport site at which the EN1 is located, not
      those five wireless sites.

### CL: Constraint Layer, responsible for logical elimination
CL may eliminate candidates only by using interpretable, stable, and reusable logical constraints
     . CL contains hard constraints. Any candidate that does not satisfy the CL rules must be
     eliminated. Before each elimination, provide a reason in the internal reasoning, but do not
     output the reasoning process in the final answer.

#### CL Rules
CL-1 Temporal-causality rule
- An alarm related to a root cause must occur before the group-outage symptoms, or at least no
     later than the first critical symptom of the group outage.
- If the key evidence for a candidate appears only after the group outage occurs, the candidate
     cannot be an independent root cause.

CL-2 Upstream-coverage rule
- An independent root cause must be located on an upstream transport path of an affected site, or
      at the endpoints of a link that constitutes the root cause.
- If a candidate is neither on any upstream path of an affected site nor capable of explaining
     the shared failure pattern among affected sites, eliminate it.

CL-3 Derivative-symptom exclusion rule
- An alarm that merely reflects the result of a service interruption does not constitute an
     independent root cause.
- For example, wireless-base-station-side alarms such as `BTS O&M LINK FAILURE`, a site going
     offline, service unreachability, or a large-scale downstream loss of connectivity are only
     derivative symptoms, rather than independent root causes, when no more direct root-cause
     evidence supports them.

CL-4 Directness-priority rule
- `System Power off` is generally a symptom rather than a root cause. If it can be traced further
      to power evidence such as `dying gasp`, `Mains Failure`, or `DC Low Voltage`, use the more
     direct upstream evidence.
- `Link Down`, `Physical Port Down`, and `ETH_LOS` are ambiguous. If there is a more direct `
     Fiber Break` alarm or explicit evidence that the peer lost power, prefer the more direct
     explanation.

CL-5 Redundancy-availability rule
- If a candidate is only a single-link or single-node failure, while the topology shows a
     complete available backup path with no evidence of abnormality, the candidate alone is
     insufficient to explain the group outage. Eliminate it, or wait until it can be evaluated
     together with another candidate.

CL-6 Shared-impact rule
- A group outage is generally not caused independently by many unrelated downstream-edge alarms.
     It is instead caused by an upstream device fault, a link fault, or a small number of
     independent faults capable of jointly explaining multiple affected sites.
- If a candidate explains only a very small number of edge symptoms and cannot explain the main
     body of the group outage, prioritize eliminating it.

CL-7 `cause_type` constraint rule
- `cause_type` must be exactly one of the following three values: `"power failure"`, `"fiber
     failure"`, or `"hardware failure"`.
- Apply the mapping rules below in descending priority order, and stop as soon as the first rule
     matches:
  - [Power failure] When any of the following evidence appears on a **transport-layer device** (
     an EN1/AN1/OPT/CN router), set `cause_type` to `power failure`:
    - `dying gasp` or `cseShutDownNotify` (direct evidence of an instantaneous device power loss)
    - `Mains Failure` or `DC Low Voltage` (power alarms from a transport device)
    - The ticket/description explicitly contains `PLN OFF`, `PLN Outage`, commercial-power
     interruption, or power loss, and the described subject is a transport device
  - [Fiber failure] When any of the following evidence appears, set `cause_type` to `fiber
     failure`:
    - `Fiber Break`, `FO Cut`, `Fiber Cut`, `Fiber/Link Break`, `partial cut`, or `cable cut`
    - The ticket/description explicitly contains "fiber cut," "fiber break," "optical-path
     interruption," "FOC," or "cable damage"
    - A transport device (EN1/AN1/OPT) reports `ETH_LOS`, `Link Down`, or `Physical Port Down`,
     and the **same device has no power alarm** (no dying gasp, Mains Failure, or PLN OFF)
    - A `Device Powered Off` alarm appears on a transport device, but there is **no dying gasp or
      Mains Failure corroboration on the same device**. Prefer interpreting this as the device
```

**Final evolved EL/CL/VL prompt (continued)**

```
      becoming unreachable because of an upstream fiber interruption, rather than as a true power
      loss
  - [Hardware failure] When any of the following evidence appears, set `cause_type` to `hardware
      failure`:
    - `SFP Faulty`, `dirty patch cord`, `patch cord not well connected`, `commcase`, a stolen
      device, `relay burned`, or `MCB tripped` (without a PLN or lightning-strike context)
- **Critical distinction rules**:
  - `Device Powered Off` alone (without dying gasp or Mains Failure) => fiber failure (the device
      is unreachable because of an upstream fiber interruption, rather than truly powered off)
  - Base-station-level alarms in the `Power_Environment` domain, including `DC Low Voltage`, `
      Fuse Blown`, and `LVD Alarm` => these are local base-station power alarms and **do not
      represent the `cause_type` of a transport-layer root cause**; do not classify the root cause
      as a power failure on this basis
  - `ETH_LOS` and `Fiber Break` occur together => fiber failure
  - `ETH_LOS` and `dying gasp`/`Mains Failure` occur together on the same transport device =>
      power failure (`ETH_LOS` is a derivative alarm of the power failure)
  - `PLN OFF` appears in the ticket description and the subject is a transport device => power
      failure (higher priority than a contemporaneous fiber alarm)
  - `dying gasp`/`Mains Failure`/`cseShutDownNotify` and `Fiber Break` occur together on the same
      transport device => **power failure** (dying gasp is direct physical evidence of power loss
      and has higher priority than Fiber Break; in this case, Fiber Break is an accompanying
      symptom of the device becoming unreachable after the power failure)
  - `SFP Faulty`/`SFP Problem`/`commcase`/`Comecase`/`relay burned`/`MCB tripped`/`stolen` and `
      ETH_LOS`/`Link Down` occur together => **hardware failure** (the link alarm is a derivative
      symptom of the hardware fault; use the direct hardware evidence)
- If a group outage has multiple independent root causes with different `cause_type` values,
      output them separately. Do not forcibly assign the same `cause_type` to all root causes.

### VL: Verdict Layer, responsible for heuristic filtering, prioritization, demotion, and merging
VL may be executed only after CL. VL does not contain hard elimination logic. Instead, it ranks,
      merges, demotes, and finalizes the surviving candidates.

#### VL Rules
VL-1 Direct-evidence priority
- A candidate with direct evidence has higher priority than a candidate supported only by an
      indirect link symptom.
- The approximate priority order is: direct evidence of power loss / explicit evidence of a fiber
      break > explicit physical-link abnormality > ordinary Link Down symptoms.

VL-2 Early-damage priority
- Among valid candidates, a candidate that occurs earlier and may have broken redundancy first
      has higher priority.

VL-3 Shared-upstream priority
- A candidate capable of jointly explaining multiple affected sites has higher priority than a
      candidate that explains only one edge site.

VL-4 More-direct-explanation priority
- If two surviving candidates can explain the same group of symptoms, preferentially retain the
      candidate that is more direct, farther upstream, and supported by stronger evidence. Demote
      the other candidate to a derivative explanation and do not output it separately.
- However, if two candidates correspond to different fault locations, different timelines, or
      different sources of direct evidence, retain them as two independent root causes even if
      both explain some of the same sites.

VL-5 Same-fault merging rule
- If multiple surviving candidates are essentially different descriptions of the same independent
      fault, they may be merged.
- After merging, retain the complete `sites` and `network_elements`, but do not incorrectly merge
      two faults that are actually independent.
- Do not merge all sites or network elements along an entire fault-propagation chain into an
      oversized candidate merely because multiple affected sites go offline simultaneously. The
      final candidate must be anchored at the true fault location.

VL-6 Conservative-underreporting rule
- The final output should be the smallest set of mutually independent root causes sufficient to
      explain the group outage.
- Do not omit an independent root cause merely for conciseness, and do not output many duplicate
      or derivative items merely to increase coverage.
- When the evidence shows that the group outage is jointly triggered by "an earlier redundancy
      fault + a later second interruption," output multiple independent root causes rather than
      retaining only the fault closest to the ticket time.

VL-7 Precise convergence of `sites` rule
- For every root cause in the final output, `sites` must represent the location at which the
      fault occurred, not the affected sites.
- Check: If `sites` for one root cause contains three or more sites, verify whether every site
      has its own independent, direct root-cause alarm. If those sites are actually victims of the
      same upstream fault (downstream sites that went offline), reduce `sites` to the true fault-
      location site: the site containing the upstream transport device or the two endpoint sites
      of the faulty link.
```

**Final evolved EL/CL/VL prompt (continued)**

```
- Typical error pattern: the model incorrectly uses all affected wireless sites in the incident
     ticket as `sites`. Correct procedure: trace upstream from these affected sites, locate their
      shared transport fault point, and use the site containing that fault point as `sites`.

## Execution Procedure
Strictly follow the reasoning procedure below:
1. Identify the affected sites and occurrence time from the ticket.
2. Trace upstream from the affected sites through the TOPO data to identify shared paths,
     critical EN1/AN1 network elements, and possible redundancy structures.
3. Use the EL rules to enumerate candidates.
4. Use the CL rules to eliminate invalid candidates one by one.
5. Apply the VL rules only to candidates that survive CL, performing prioritization, demotion,
     and merging.
6. Output the final set of independent root causes.

## Supplementary Operational Knowledge
1. Group outages are often triggered by two types of problems:
   - A router or transport device loses power or goes offline.
   - A physical interruption occurs between routers or on a transport link, especially when
     redundant paths are damaged in succession.
2. A normal path endpoint with no alarm does not mean that the entire transport path is available
     . Link continuity must be checked.
3. An early alarm may precede the ticket by hours or even days, but may still be part of the true
      root cause.
4. AN1 is generally closer to the aggregation layer than EN1 and is more likely to create a
     shared impact scope. However, do not mechanically select AN1; evidence is still required.
5. For a link alarm, combine Alarm_Location_Info, the peer relationship, and the upstream/
     downstream path. Do not draw a conclusion from the alarm name alone. When `Device Powered
     Off` appears on a transport NE, do not directly classify it as a power failure. When a fiber
      is cut, the network-management system may likewise report that a device is "unreachable/
     powered off." In this case, first look for a dying gasp or Mains Failure alarm on the same
     device as direct evidence of a power failure; if none exists, favor a fiber failure.
9. Base-station alarms in the Power_Environment domain (including DC Low Voltage, Fuse Blown, and
      LVD Alarm) reflect the base station's local power state and are not evidence for the `
     cause_type` of a transport-layer root cause. When determining the `cause_type` of a group
     outage, focus on alarms from transport devices (EN1/AN1/OPT), rather than base-station-level
      power alarms.
6. When multiple EN1 elements along an entire chain topology simultaneously report Link Down /
     ETH_LOS, locate the "most upstream break": identify the link segment closest to AN1 that has
      direct fault evidence. Do not list every node along the chain in `sites`.
7. If an EN1 router connects to AN1 in two directions (primary/backup) and there is independent
     interruption evidence on both paths, these interruptions are generally two independent root
     causes and must be output separately.
8. A stolen device or hardware damage (relay burned or MCB tripped) is a `hardware failure`, not
     a `power failure` or `fiber failure`.

## Output Requirements
Your final output must meet all requirements below:
- The output must contain one JSON array and nothing else.
- Each element in the array represents one independent root cause.
- Every element must contain exactly the following fields:
  - `sites`: array
  - `network_elements`: array
  - `cause_type`: string
- Do not output explanations, prefixes, suffixes, Markdown, or code-block markers.
- Do not output chain-of-thought, rule IDs, or extra fields.

## Detailed Output-Field Requirements
1. `sites`
- `sites` denotes the site corresponding to the location where the root cause actually occurred.
- For a device root cause (for example, an EN1 router loses power), specify the site at which the
      device is located. Do not specify every downstream base-station site affected by that
     device.
- For a link root cause (for example, a fiber break between two EN1 elements), specify the sites
     at both ends of the link. If the evidence indicates that multiple link segments of the same
     type failed together, the corresponding endpoint sites may be specified.
- If a group outage has multiple independent root causes, specify the `sites` for each root cause
      independently, covering only the fault-location sites directly related to that root cause.

2. `network_elements`
- Specify the names of the network elements directly related to the fault.
- Even if the reference-answer style emphasizes sites and de-emphasizes network elements, extract
      the network elements whenever possible. If they genuinely cannot be determined reliably
     from the evidence, an empty array is allowed.

3. `cause_type`
- Select exactly one of the three allowed values; do not output any other category.

## Important Prohibitions
- Do not directly treat a derivative link-failure alarm from a wireless base station as an
     independent root cause.
- Do not treat a recovery-related or accompanying alarm after the group outage as a root cause.
```

**Final evolved EL/CL/VL prompt (continued)**

```
- Do not repeatedly output multiple symptoms that can explain one another as the same fault-
    propagation chain.
- Do not omit multiple mutually independent root causes when each is supported by evidence.

## Example Output Format
[
  {
    "sites": ["Transport Site A", "Transport Site B"],
    "network_elements": ["Aggregation NE X", "Aggregation NE Y"],
    "cause_type": "fiber failure"
  },
  {
    "sites": ["Transport Site C"],
    "network_elements": ["Aggregation NE Z"],
    "cause_type": "power failure"
  }
]
```

# D COMPLETE TOOL AND VERIFIER SPECIFICATIONS

This section gives the complete operational contract of the three domain commands used by AVERT: `filter_alarms`, `restore_paths`, and `diagnose`. In the evaluated harness, the model was registered with one generic `bash` function whose sole required argument was a string named `command`; the three deterministic capabilities were exposed as subcommands of

```
python -m rca_swe_agent.tools.cli <command> [incident_json] [options]
```

The incident file was normally bound out of band through `RCA_INCIDENT_JSON`, so the model did not generate its path. The command-line interface writes a human-readable report to standard output. For unambiguous reproduction, the schemas below specify the equivalent JSON request and the JSON-serializable object immediately before report formatting. Executable labels are shown as JSON Unicode escapes where necessary; for example, `"\u5149\u7ea4\u95ee\u9898"` is the label for a fiber fault. The English descriptions below are faithful translations of the Chinese descriptions supplied to the model; command names, option names, matching rules, and stopping conditions are verbatim implementation behavior.

## D.1 ALARM RETRIEVAL: FILTER_ALARMS

**Operational description.** "Filter alarm records from the raw alarm table by network-element name (substring match) or site ID (exact match)." The exact CLI forms are

```
python -m rca_swe_agent.tools.cli filter_alarms \
  [incident_json] --names NE_A,NE_B
python -m rca_swe_agent.tools.cli filter_alarms \
  [incident_json] --site-ids SITE_A,SITE_B
```

The two selectors may be supplied together and are combined by union. At least one non-empty selector is required.

**Normalized input schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "incident_json": {"type": "string", "minLength": 1},
    "names": {
      "type": "array", "items": {"type": "string", "minLength": 1},
      "uniqueItems": true, "default": []
    },
    "site_ids": {
      "type": "array", "items": {"type": "string", "minLength": 1},
      "uniqueItems": true, "default": []
    }
  },
  "anyOf": [
    {"required": ["names"], "properties": {"names": {"minItems": 1}}},
    {"required": ["site_ids"], "properties": {"site_ids": {"minItems": 1}}}
  ],
  "additionalProperties": false
}
```

`incident_json` is optional only when `RCA_INCIDENT_JSON` is set by the harness.

**Normalized output schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "required": ["table", "rows", "matched_count", "depth", "field",
               "seen_keys_out"],
  "properties": {
    "table": {"type": "string"},
    "rows": {"type": "array", "items": {"type": "string"}},
    "matched_count": {"type": "integer", "minimum": 0},
    "depth": {"const": "high"},
    "field": {"const": "\u544a\u8b66\u539f\u59cb\u6570\u636e_\u5fc5\u987b"},
    "seen_keys_out": {
      "type": "array",
      "items": {
        "type": "array", "minItems": 3, "maxItems": 3,
        "items": {"type": "string"}
```

```
      }
    }
  },
  "additionalProperties": false
}
```

**Fields, time range, deduplication, and order.** The `names` selector is matched against `Alarm_Source` using case-insensitive bidirectional substring matching: a row matches when the query is a substring of the source or the source is a substring of the query. The `site_ids` selector is matched case-insensitively but otherwise exactly against `Site_ID`. No other column is used as a predicate. All columns in the raw table are retained, including `Alarm_Name`, `First_Occurred_On`, `Cleared_On`, `Alarm_Source`, `Domain`, `Site_ID`, `Alarm_Severity`, `Alarm_Location_Info`, `Device_Type`, `Network_Type`, `Vendor`, work-order ID, ticket ID, and alarm-coding ID when present.

The search range is the complete per-incident raw table identified by the `field` constant in the schema; `filter_alarms` applies no additional lower or upper timestamp bound and does not filter on `Cleared_On`. Rows already present in the processed alarm table, rows supplied in `seen_keys`, and duplicates within the current result are removed using the key (`Alarm_Source`, `First_Occurred_On`, `Alarm_Name`). The remaining rows preserve their order in the raw table; the tool performs no chronological or severity re-sorting. Consequently, an empty result means "no new row matched these selectors," not "the element had no alarm."

### D.2 Path Restoration: `RESTORE_PATHS`

**Operational description.** "Output the recovered service paths from each affected site to an aggregation node (AN1/CN1/CN2), together with convergence nodes shared by site paths." The exact CLI forms are

```
python -m rca_swe_agent.tools.cli restore_paths [incident_json]
python -m rca_swe_agent.tools.cli restore_paths \
  [incident_json] --sites SITE_A,SITE_B
```

Omitting `--sites` returns all ticket-listed affected sites; supplying it only changes the displayed subset and does not change the graph used for restoration.

**Normalized input schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "incident_json": {"type": "string", "minLength": 1},
    "sites": {
      "type": "array", "items": {"type": "string", "minLength": 1},
      "uniqueItems": true, "default": []
    }
  },
  "additionalProperties": false
}
```

**Normalized output schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$defs": {
    "path": {"type": "array", "minItems": 2,
             "items": {"type": "string", "minLength": 1}}
  },
  "type": "object",
  "required": ["incident_id", "agg_nodes", "agg_sites",
               "affected_sites", "paths"],
  "properties": {
    "incident_id": {"type": "string"},
    "agg_nodes": {"type": "array", "items": {"type": "string"}},
    "agg_sites": {
      "type": "object",
      "additionalProperties": {
        "type": "array", "items": {"type": "string"}
      }
    },
    "affected_sites": {"type": "array", "items": {"type": "string"}},
    "paths": {
```

```
      "type": "object",
      "additionalProperties": {
        "type": "array", "items": {"$ref": "#/$defs/path"}
      }
    }
  },
  "additionalProperties": false
}
```

**Restoration and stopping conditions.** The tool parses the topology as a directed adjacency list and merges site-to-element associations from the power-information block, the incident ticket, topology-name prefixes, and both processed and raw alarm tables. For every network element associated with an affected site, breadth-first search enumerates all equal-length shortest directed paths to a node whose name contains `AN1`, `CN1`, or `CN2`. A path is never allowed to revisit a node. Search along a branch stops when (i) an aggregation/core node is reached, (ii) the path would exceed 12 nodes, or (iii) the breadth-first frontier is longer than the best aggregation-reaching path already found. Thus "all paths" in the tool output means all recovered equal-length shortest paths under this bound, not all simple paths in the physical graph.

After search, a same-site prefix is removed so that each returned path begins at the last element belonging to the affected site. Identical post-pruning paths are deduplicated. If the incident contains a precomputed path in `\u8ba1\u7b97\u597d\u7684TOPO\u6570\u636e`, it is appended when its exact node sequence is not already present. Finally, a convergence node is any node reached through at least two distinct immediate predecessors across the restored paths (a path start contributes a virtual predecessor); `agg_sites` records the affected sites whose paths contain each such node.

### D.3 TOPOLOGY-IMPACT VERIFIER: `DIAGNOSE`

**Operational description.** "Given root-cause triples (sites, network_elements, cause_type), compute which service paths and affected sites would be blocked by the corresponding node or link failures." The exact multi-candidate form is

```
python -m rca_swe_agent.tools.cli diagnose \
  [incident_json] --fault '<JSON array of root-cause triples>'
```

The executable also accepts `--sites`, `--nes`, and `--cause` for one candidate. The benchmark uses the fiber, power, and hardware labels; the verifier additionally retains a configuration label in its executable interface.

**Normalized input schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$defs": {
    "fault": {
      "type": "object",
      "required": ["sites", "cause_type"],
      "properties": {
        "sites": {"type": "array", "minItems": 1,
                  "items": {"type": "string", "minLength": 1}},
        "network_elements": {"type": "array", "default": [],
                             "items": {"type": "string", "minLength": 1}},
        "cause_type": {
          "enum": ["\u5149\u7ea4\u95ee\u9898", "\u4f9b\u7535\u95ee\u9898",
                   "\u786c\u4ef6\u95ee\u9898", "\u914d\u7f6e\u95ee\u9898"]
        }
      },
      "additionalProperties": false
    }
  },
  "type": "object",
  "required": ["faults"],
  "properties": {
    "incident_json": {"type": "string", "minLength": 1},
    "faults": {"type": "array", "items": {"$ref": "#/$defs/fault"}}
  },
  "additionalProperties": false
}
```

**Normalized successful-output schema.**

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$defs": {
    "path": {"type": "array", "items": {"type": "string"}},
    "site_result": {
      "type": "object",
      "required": ["total", "blocked", "surviving", "affected"],
      "properties": {
        "total": {"type": "integer", "minimum": 0},
        "blocked": {"type": "array", "items": {"$ref": "#/$defs/path"}},
        "surviving": {"type": "array", "items": {"$ref": "#/$defs/path"}},
        "affected": {"type": "boolean"}
      },
      "additionalProperties": false
    }
  },
  "type": "object",
  "required": ["failed_nodes", "failed_edges", "site_results",
               "affected_sites", "surviving_sites", "no_path_sites"],
  "properties": {
    "failed_nodes": {"type": "array", "items": {"type": "string"}},
    "failed_edges": {"type": "array", "items": {
      "type": "array", "minItems": 2, "maxItems": 2,
      "items": {"type": "string"}
    }},
    "site_results": {"type": "object",
                     "additionalProperties": {"$ref": "#/$defs/site_result"}},
    "affected_sites": {"type": "array", "items": {"type": "string"}},
    "surviving_sites": {"type": "array", "items": {"type": "string"}},
    "no_path_sites": {"type": "array", "items": {"type": "string"}}
  },
  "additionalProperties": false
}
```

Validation errors are emitted before this object is constructed; the CLI prints each error and returns without simulating impact. Duplicate-candidate warnings are emitted before an otherwise valid report.

**Projection from predictions to failures.** Let network elements be consumed in consecutive pairs $(n_1, n_2), (n_3, n_4), \ldots$. The deterministic mapping is:

| Cause | Structural mode | Failure inserted into the simulation |
|---|---|---|
| Fiber | any declared sites | Every consecutive element pair becomes a failed undirected link; at least two elements, an even count, distinct endpoints, and direct adjacency are required. |
| Power | exactly one site | All network elements associated with the site become failed nodes. A supplied element list does not narrow this site-wide expansion. |
| Hardware or configuration | one site | The listed elements become failed nodes; if the list is empty, all elements associated with the site become failed nodes. |
| Hardware or configuration | two sites | Every consecutive element pair becomes a failed undirected link; paired endpoints must be distinct and directly adjacent. |

An element mentioned in a candidate must occur in either the parsed topology or the merged site-to-element map. For a link failure, adjacency may appear in either direction in the input topology. A fiber candidate with an odd number of elements, a self-pair, an unknown element, a non-adjacent pair, a multi-site power candidate, or a hardware/configuration candidate with a site count other than one or two is structurally invalid and is not simulated.

**All-paths-cut criterion.** The following pseudocode is equivalent to the path-impact calculation. Failed links are stored in both directions for membership tests and canonicalized only for display.

```
function PATH_BLOCKED(path, failed_nodes, failed_links):
    if any node in path is in failed_nodes: return true
    if any consecutive pair (path[i], path[i+1]) is in failed_links:
        return true
    return false

function VERIFY(affected_sites, paths_by_site, predicted_faults):
    failed_nodes, failed_links = PROJECT(predicted_faults)
    for site in affected_sites:
        paths = paths_by_site[site]
        if paths is empty:
```

```
            result[site] = {total: 0, blocked: [], surviving: [],
                            affected: true}
            continue
        blocked   = [p for p in paths if PATH_BLOCKED(p, failed_nodes,
                                                      failed_links)]
        surviving = [p for p in paths if p not in blocked]
        result[site].affected = (length(surviving) == 0)
    return result
```

For every site with at least one restored path, the criterion is therefore exactly

$$\text{covered}(s) \iff \forall p \in \mathcal{P}_s : \big(p \cap F_V \neq \emptyset\big) \ \vee \ \big(E(p) \cap F_E \neq \emptyset\big).$$

The executable uses the no-surviving-path convention shown above when $\mathcal{P}_s = \emptyset$. Because that boundary case contains no positive topology evidence, the operational protocol treats the explicit "unable to restore path" warning from `restore_paths` as *unverifiable*, rather than as a sufficient reason to accept a diagnosis. In all cases, a topology pass is necessary but not sufficient: alarm evidence, temporal precedence, and causal admissibility remain governed by CL and VL.

### D.4 Validation Against Operational Business Paths

Both `restore_paths` and the topology-impact verifier passed an implementation-level correctness check against operator-provided records of the actual business paths used in the production network. This validation was performed before the tools were used in the reported experiments.

**Path-restoration validation.** The path-restoration correctness check comprised 131 reviewed business-path samples. `restore_paths` correctly reconstructed all 131 samples, yielding 100% path-restoration accuracy (131/131). Each recovered site-to-aggregation sequence was compared with the corresponding operational business path. The check covered the affected-site endpoint, terminating aggregation node, hop order, adjacency of consecutive network elements, alternative branches, and shared convergence points. A reconstruction was counted as correct only when every reported hop formed a real connection and the recovered path preserved the deployed service-routing sequence; discrepancies were resolved in the deterministic parsing and site-to-element mapping logic rather than accepted as approximate matches.

**Verifier validation.** The topology verifier was validated on these reviewed business paths by replaying operator-confirmed node and link failures. For every replay, its per-path `blocked`/`surviving` classification and its site-level "all business paths are cut" decision were checked against the operational outage scope. This jointly tests the prediction-to-failure projection and the all-paths-cut calculation, rather than validating only that the code runs. The business-path records used for this implementation check were not exposed to the RCA model, were not supplied as incident-time evidence, and did not enter Rule Evolution feedback or test-time decisions.

### D.5 Exceptions and Boundary Cases

Table 6: Deterministic handling of tool failures and boundary cases.

| Condition | Handling |
|---|---|
| Missing incident path, unreadable file, malformed JSON, or malformed required table | The command fails (or prints an explicit error before returning); no partial output is treated as evidence. The incident-level harness may restart the task, up to the three retries specified in Appendix F. |
| No alarm selector | `filter_alarms` prints an argument error and performs no search. |
| No matching alarm row | Valid empty retrieval: `matched_count`=0 and the CLI prints "no matching alarm." This is not evidence that no fault exists. |
| Requested site not in ticket | `restore_paths` prints a warning and omits that site from the displayed path map. |
| No recoverable path | `restore_paths` returns an empty path list and an explicit warning. The case is treated as unverifiable at the protocol level and must not alone certify coverage. |
| Repeated selector, path, node, link, or fault group | Selector duplicates are removed by set construction; repeated nodes are forbidden within a recovered path; identical paths are deduplicated by their complete node sequence. Repeated failed-node/link membership is idempotent in simulation. An exactly repeated fault triple produces a warning but does not abort. |
| Element cannot be matched | Alarm retrieval performs no fuzzy fallback and returns an empty match. In `diagnose`, an element absent from both topology and the merged site-element map is a structural error; simulation stops until the name is corrected. |
| Invalid link endpoint pair | Self-pairs, odd fiber element counts, and endpoints that are not directly adjacent in either topology direction are errors; no impact report is generated for the candidate set. |

# E FULL RULE EVOLUTION PROTOCOL

This section specifies the complete offline protocol used for Responsibility-Routed Rule Evolution. The evolution driver is Codex (GPT-5.4), while every incident trajectory is produced by the Qwen3.5-27B RCA agent. Evolution uses only the 51 earlier training incidents. We run 15 rounds, select the eight training incidents with the lowest RCA Scores per round, and generate exactly one candidate rule state per round. The 80 later test incidents are unavailable to Codex and are not used for feedback, routing, acceptance, rollback, or final-state selection.

## E.1 EVOLUTION META-PROMPT

The following is the complete instruction template given to Codex. Braced uppercase tokens are deterministic runtime substitutions rather than model-written text. All file paths exposed through the capsule indices are read-only.

**Codex evolution meta-prompt**

```
You are the TRAJECTORY-ROUTED LOCAL REPAIRER for an RCA-Agent EL-CL-VL rule system.

The fixed prompt scaffold already fuses the RCA task, inputs, candidate structure,
EL->CL->VL execution order, and JSON contract. There is no competing legacy
workflow.

Inspect all supplied ordinary Qwen trajectories before choosing exactly one target
layer:
- EL owns candidate enumeration, evidence coverage, and recall. It must not hard
  eliminate candidates or make the final ranking.
- CL owns conservative hard elimination and precision. Every rejection needs a
  reusable causal reason.
- VL owns only post-CL soft ranking, normalization, merging, and final-verdict
  selection.

The metric route attached to each capsule is only a pre-routing hint. EL means the
weighted recall deficit dominates; CL means the weighted precision deficit
dominates; AMBIGUOUS means those deficits are balanced. AMBIGUOUS is never evidence
for VL. Select VL only when the complete trajectory shows that viable candidates
were enumerated and conservatively filtered, and the first material error occurred
during post-CL ranking, normalization, merging, or final-set selection.

For every ordinary capsule, add one routing_analysis item grounded in the first
wrong Qwen turn. Record the metric hint, your trajectory-selected layer, concrete
turn evidence, why that layer owns the failure, and routing confidence. Then choose
target_layer as the single layer whose trajectory-grounded failure cluster offers
the highest-impact reusable repair. Consider recent target_layer history so a
persistent severe failure is not ignored merely because another layer has a small
plurality in the metric hints.

Do not memorize incident IDs, exact answers, split names, or scores in rule edits.
Use read-only file/shell tools to inspect every bundle listed in ROUTED TRAINING
ERROR CAPSULE INDEX. Access only those listed bundle paths; do not browse the
repository, dataset directories, experiment manifests, or any unlisted path. The
benchmark test set is unavailable. The complete compiled system prompt must be at
most 12,000 characters; the current character count is supplied below.

Return only one structured rule-diff candidate in rule_operations; never return or
repeat the complete prompt or complete EL/CL/VL arrays. Every operation must stay in
the chosen target_layer:
- add: use a new target-prefixed rule_id, non-empty text, and after_rule_id
  ("__START__" for the start or "" for the end).
- delete: use an existing target-layer rule_id with empty text and after_rule_id.
- rewrite: use an existing target-layer rule_id with replacement text and empty
  after_rule_id.
- reorder: use an existing target-layer rule_id with empty text and the same
  positioning convention as add.

Cross-layer edits, move operations, and flat-prompt edits are forbidden. Every
operation reason must cite a concrete Qwen reasoning failure, explain target-layer
ownership, expected effect, and regression risk. The top-level rationale must
explain the trajectory-grounded routing decision, local repair, unchanged layers,
and likely side effects.

Use rejection_analysis to compare the most recent rejected candidate with the
restored incumbent, explain why that local edit failed, and name the edit pattern
that must not be repeated. On the first round, set rejection_analysis to
"No prior rejected candidate.".

CURRENT EL-CL-VL RULE SYSTEM
```

**Codex evolution meta-prompt (continued)**

```
{{CURRENT_RULE_SYSTEM_JSON}}

ROUTED TRAINING ERROR CAPSULE INDEX (all selected cases; open files with tools)
{{TRAINING_FEEDBACK_JSON}}

TRAINING TRAJECTORY-FIRST DIAGNOSIS
Before designing any edit, use tools to inspect every ordinary training bundle
listed in the capsule index. For each bundle, read conversation_log.json from start
to finish, including the model's thinking, actions, tool calls, observations, and
revisions. Also read trajectory.json when present. Identify the first turn where
the model made an unsupported inference, misread evidence, chose an unhelpful tool
action, stopped searching too early, or failed to consider an available clue.

Only after reconstructing that reasoning process may you consult result.md,
prediction/ground-truth files, evaluator output, or aggregate scores. Do not
diagnose an error merely by comparing the final answer with the ground truth or by
guessing from statistics. Ground every proposed rule change in a concrete reasoning
failure or missed decision visible in the Qwen trajectory. If trajectory.json is
absent, use the complete conversation log and state that limitation in the
rationale.

REGRESSION COMPARISON CAPSULE INDEX (open both sides with tools)
{{REGRESSION_COMPARISON_JSON}}

PROCESS-FIRST REGRESSION DIAGNOSIS
If the index contains cases, inspect every case before designing an edit. Read both
complete conversation_log.json files from start to finish, plus both available
trajectory.json files and prompt_change.diff. Compare the incumbent and rejected
candidate thinking, actions, tool calls, and tool results before consulting their
final answers or evaluator files. Locate the first incumbent turn relevant to the
divergence and the first candidate turn where reasoning became wrong, unsupported,
over-broad, or failed to pursue available evidence.

For each case, add one trajectory_analysis item recording the relevant incumbent
turn, candidate first wrong turn, exact reasoning failure, evidence available but
missed or misread, tool action or missing action that changed the conclusion, the
implicated rejected prompt edit, and a reusable safe correction. Do not infer the
failure only from final answers, score deltas, evaluator output, or aggregate
statistics. Incident IDs may be copied into trajectory_analysis for audit, but must
never be encoded into the evolved rules. If the index has no cases, return an empty
trajectory_analysis array. This analysis is an audit aid, not a separate mutation
acceptance gate.

INCUMBENT TRAINING AGGREGATE ONLY
{{INCUMBENT_TRAINING_AGGREGATE_JSON}}

CURRENT COMPILED-PROMPT CHARACTER COUNT
{{CURRENT_PROMPT_CHARACTER_COUNT}}

PRIOR ROUTED ACCEPT/REJECT HISTORY (at most the six most recent rounds)
{{RECENT_HISTORY_JSON}}

MOST RECENT REJECTED ROUTED CANDIDATE (complete state and metrics)
{{LATEST_REJECTED_CANDIDATE_JSON_OR_NULL}}

If the previous candidate was rejected or rolled back, CURRENT EL-CL-VL RULE SYSTEM
is the restored incumbent. Do not repeat the rejected local edit shown in history;
derive a different repair while remaining strictly inside the newly selected layer.
```

### E.2 Evaluator Feedback and Error Capsules

For each training incident, evaluator v2 returns the following object. All numeric fields lie in $[0, 1]$.

```
{
  "match_level": {
    "precision": <number>, "recall": <number>, "f1": <number>
  },
  "field_level": {
    "sites": {
      "precision": <number>, "recall": <number>, "f1": <number>
    },
    "network_elements": {
      "precision": <number>, "recall": <number>, "f1": <number>
    },
    "cause_type": {
      "precision": <number>, "recall": <number>, "f1": <number>
    }
  },
  "rca_score": <number>
}
```

Writing $F_m$ for match-level F1, the evaluator's RCA Score is

$$S_{\text{RCA}} = F_m \left(0.5F_{\text{sites}} + 0.3F_{\text{NE}} + 0.2F_{\text{type}}\right).$$

At the beginning of every round, incidents are sorted by $(S_{\text{RCA}}, \texttt{incident_id})$ in ascending lexicographic order and the first eight are selected. Thus ties are deterministic. The following JSON is the exact logical feedback template; angle-bracketed values are filled by the driver.

```
{
  "evaluator": "v2",
  "selection": "lowest training RCA Score",
  "routing_strategy": "trajectory_routed_local",
  "metric_hint_semantics": {
    "EL": "weighted recall deficit dominates",
    "CL": "weighted precision deficit dominates",
    "AMBIGUOUS": "metric deficits are balanced; not evidence for VL"
  },
  "metric_priority_hint": "<EL|CL|AMBIGUOUS>",
  "routing_weights": {
    "EL": <number>, "CL": <number>, "AMBIGUOUS": <number>
  },
  "capsule_root": "<absolute read-only directory>",
  "capsules": [
    {
      "incident_id": "<id>",
      "rca_score": <number>,
      "route": "<EL|CL|AMBIGUOUS>",
      "routing_diagnostics": {
        "precision_deficit": <number>,
        "recall_deficit": <number>,
        "deficit_margin": <number>
      },
      "bundle_dir": "<absolute read-only directory>",
      "files": [
        "incident_data.json", "conversation_log.json",
        "evaluation_v2.json", "prediction_and_ground_truth.json",
        "capsule_meta.json", "<optional runtime files>"
      ]
    }
  ],
  "regression_comparison_capsules": {
    "selection": "most negative rejected-candidate train RCA Score deltas",
    "source_round": <integer>,
    "source_candidate": 1,
    "max_cases": 4,
    "capsule_root": "<absolute read-only directory>",
    "cases": [
      {
        "incident_id": "<id>",
        "incumbent_rca_score": <number>,
        "candidate_rca_score": <number>,
        "score_delta": <candidate_rca_score-minus-incumbent_rca_score>,
        "bundle_dir": "<absolute read-only directory>",
        "files": {
          "shared": ["incident_data.json", "comparison_meta.json",
                     "prompt_change.diff"],
          "incumbent": ["conversation_log.json", "evaluation_v2.json",
                        "prediction_and_ground_truth.json", "<optional files>"],
          "rejected_candidate": ["conversation_log.json",
                                 "evaluation_v2.json",
                                 "prediction_and_ground_truth.json",
                                 "<optional files>"]
        }
      }
    ]
  }
}
```

The comparison field is omitted in round 1 and after an accepted round. After an immediate rejection, all negative training RCA Score deltas are sorted by $(\Delta S_{\text{RCA},i}, \texttt{incident_id})$; the four most negative cases are paired with the incumbent and rejected-candidate traces for the next round. Optional runtime files are `trajectory.json`, `result.md`, `stream.log`, `error.txt`, `extraction_failure.json`, and retry-specific error logs. A missing `conversation_log.json` for any selected non-mock case is a fatal capsule-construction error; it is never replaced by final-answer-only feedback.

The metric hint is computed from precision and recall deficits

$$D_P = 1 - P_m + 0.5(1 - P_{\text{sites}}) + 0.3(1 - P_{\text{NE}}) + 0.2(1 - P_{\text{type}}),$$
$$D_R = 1 - R_m + 0.5(1 - R_{\text{sites}}) + 0.3(1 - R_{\text{NE}}) + 0.2(1 - R_{\text{type}}).$$

The per-case hint is EL if $D_R > D_P + 0.20$, CL if $D_P > D_R + 0.20$, and `AMBIGUOUS` otherwise. Its routing weight is $\max(0.01, 1 - R)$. The batch-level `metric_priority_hint` is

the unique larger accumulated weight among EL and CL; a tie or absence of confident hints yields `AMBIGUOUS`. These values are pre-routing diagnostics only: Codex must inspect trajectories before selecting the responsible layer.

### E.3 Trajectory Format and Inspection Order

The complete incident trace has the following JSON shape. `trajectory.json` contains the same `info`, `messages`, and `conversation_log` fields; `conversation_log.json` additionally records final extraction and evaluation.

```
{
  "info": {"<run metadata>": "<value>"},
  "messages": [
    {"role": "<system|user|assistant|tool>", "content": "<text>"}
  ],
  "conversation_log": [
    {
      "turn": <positive integer>,
      "content": "<assistant content or serialized tool request>",
      "thinking_content": "<model reasoning for this turn>",
      "tool_calls": [
        {
          "name": "bash",
          "arguments": {"command": "<command string>"},
          "result": "<complete tool observation>",
          "elapsed": <seconds>
        }
      ],
      "finish_reason": "<provider finish reason>",
      "tokens": {"<token accounting>": <number>}
    }
  ],
  "root_cause_extraction": {
    "phase1_predicted": ["<optional first-stage root-cause objects>"],
    "predicted": ["<final root-cause objects>"],
    "ground_truth": ["<reference root-cause objects>"],
    "evaluation": {"<evaluator-v2 object>": "<as specified above>"}
  }
}
```

Codex reads every selected `conversation_log.json` from the first to the last turn, including reasoning, actions, tool calls, observations, and revisions; it also reads `trajectory.json` when present. Only then may it inspect `result.md`, the prediction/ground-truth pair, evaluator output, or aggregate score. The diagnosis must identify the first material error—unsupported inference, misread evidence, unhelpful or missing tool action, premature stopping, or ignored clue. If `trajectory.json` is absent but the complete conversation log exists, Codex proceeds and states that limitation in its rationale.

### E.4 Responsible-Layer Selection and Candidate Schema

Table 7: Trajectory-grounded responsible-layer criteria.

| Layer | Select when the first material error is | Exclusion / ownership boundary |
|---|---|---|
| EL | A viable cause was never enumerated because evidence coverage, candidate construction, or recall-oriented search failed. | EL may broaden evidence collection and candidate formation, but it may neither hard-eliminate a candidate nor choose the final answer. |
| CL | A candidate was incorrectly admitted or rejected by a reusable causal constraint, temporal test, topology test, or evidence-admissibility decision. | CL owns conservative hard elimination and precision. Every rejection must have an explicit reusable causal reason; ranking or merging surviving candidates is not CL work. |
| VL | Enumeration and conservative filtering were both adequate, but ranking, normalization, merging, deduplication, or final-set selection was wrong. | VL is selected only when the first material error occurs after CL. A balanced metric hint alone is never evidence for VL. |

Codex emits one `routing_analysis` entry for every ordinary capsule. It then chooses the single layer whose cluster of first-error diagnoses offers the highest-impact reusable repair, while consid-

ering recent target-layer history so that a persistent severe failure is not hidden by a small plurality of metric hints. The required response is:

```
{
  "rejection_analysis": "<why the latest rejected edit failed, or the
                         exact first-round sentinel>",
  "trajectory_analysis": [
    {
      "incident_id": "<id>",
      "incumbent_relevant_turn": "<turn and evidence>",
      "candidate_first_wrong_turn": "<turn and evidence>",
      "reasoning_failure": "<failure>",
      "missed_evidence": "<available evidence>",
      "tool_action_effect": "<effect of action or omission>",
      "implicated_prompt_edit": "<rejected edit>",
      "safe_correction": "<reusable correction>"
    }
  ],
  "routing_analysis": [
    {
      "incident_id": "<id>",
      "metric_hint": "<EL|CL|AMBIGUOUS>",
      "first_wrong_turn": "<turn-grounded description>",
      "selected_layer": "<EL|CL|VL>",
      "evidence": "<concrete trace evidence>",
      "layer_reason": "<ownership explanation>",
      "confidence": "<high|medium|low>"
    }
  ],
  "rationale": "<routing, local repair, unchanged layers, side effects>",
  "target_layer": "<EL|CL|VL>",
  "rule_operations": [
    {
      "op": "<add|delete|rewrite|reorder>",
      "rule_id": "<target-layer rule id>",
      "layer": "<same value as target_layer>",
      "text": "<new text for add/rewrite; otherwise empty>",
      "after_rule_id": "<anchor for add/reorder; otherwise empty>",
      "reason": "<trace failure, ownership, expected effect, risk>"
    }
  ]
}
```

All listed top-level fields are required, unknown fields are rejected, and `routing_analysis` and `rule_operations` must be non-empty. Operations are applied sequentially. New IDs must carry the target-layer prefix; referenced IDs and anchors must exist at the point of use. Cross-layer edits, moves, full-array replacement, empty edit sets, invalid anchors, and a compiled prompt longer than 12,000 characters are rejected before evaluation. A generation or validation failure may be retried up to three times, but these attempts repair the same single candidate slot and do not increase the one-candidate-per-round budget.

### E.5 Round Schedule, Acceptance, and Rollback

```
R = initial EL/CL/VL rule state
history = []
for round in 1..15:
    incumbent_scores = EVALUATE(R, all 51 training incidents)
    ordinary = LOWEST_8(incumbent_scores, key=(rca_score, incident_id))
    feedback = BUILD_CAPSULES(ordinary, complete traces and evaluator files)

    if the immediately preceding round rejected its candidate:
        deltas = rejected_scores - incumbent_scores_for_that_round
        feedback.regression_comparison_capsules =
            LOWEST_4(deltas restricted to deltas < 0,
                     key=(delta, incident_id))

    response = CODEX(meta_prompt(R, feedback, last_6(history)))
    VALIDATE_SINGLE_LAYER_DIFF(response)
    C = APPLY_OPERATIONS_TO_RESTORED_INCUMBENT(R, response.rule_operations)
    candidate_scores = EVALUATE(C, all 51 training incidents)

    mean_delta = mean(candidate_scores) - mean(incumbent_scores)
    regression_rate = count_i(candidate_scores[i] <
                              incumbent_scores[i] - 0.05) / 51
    if mean_delta >= 0.005 and regression_rate <= 0.35:
        R = C
        history.append(ACCEPT)
    else:
        R = R
        history.append(ROLLBACK)

freeze R before any evaluation on the 80 later test incidents
```

The two gates are conjunctive. RCA Score is stored on the unit interval, so $0.005$ is 0.5 percentage points. An incident is a material regression only when its candidate RCA Score is strictly more than 0.05 below the incumbent; a rate exactly equal to 35% passes. On rejection, the next round starts from the unchanged incumbent rather than the rejected state, and the rejected state, rule diff, metrics, and paired regression trajectories remain available only as contrastive history. The final accepted state is frozen before test evaluation; in the reported responsibility-routed run, that state was reached at round 11.

# F Complete Experimental Configuration

This section records the exact model identifiers visible to the experiment harness, their serving paths, request-level decoding settings, agent budgets, and the retained local deployment environment. For provider-hosted models, the identifiers below are API routing names: the providers did not expose a local checkpoint, weight revision, or backend commit hash. We therefore do not reinterpret an API route as a checkpoint version.

## F.1 Backbones, Checkpoints, and Serving

Table 8: Backbone identifiers or local checkpoints and serving modes.

| Model | Backbone / checkpoint | Serving mode |
|---|---|---|
| DeepSeek v3.2 | API route `deepseek-v3.2`; no local checkpoint or provider backend revision/hash was exposed. | Anonymous OpenAI-compatible API gateway; SSE streaming. |
| GLM 5.1 | API route `glm-5.1`; no local checkpoint was exposed. | Anonymous OpenAI-compatible API gateway; SSE streaming. |
| GPT-5.4 | API route `gpt-5.4`; no local checkpoint was exposed. | Anonymous OpenAI-compatible API gateway; SSE streaming. |
| Claude Opus 4.6 | API route `claude-opus-4-6`; no local checkpoint was exposed. | Anonymous OpenAI-compatible API gateway; SSE streaming. |
| Gemma 4 12B IT | `/data/anon-user0/models/gemma-4-12B-it`; local original safetensors checkpoint. | `api_server_v2.py` with vLLM; four independent TP=1 instances on GPUs 0–3, ports 8000–8003. |
| Gemma 4 31B IT | `/data/anon-user0/models/gemma-4-31B-it`; local original safetensors checkpoint with index and weight shards. | `api_server_v2.py` with vLLM; two TP=2 instances on GPU groups [0,1] and [2,3], ports 8000–8001. |
| Qwen3.5-9B | `/data/anon-user0/models/Qwen3.5-9B`; local checkpoint. | `api_server_v2.py` with vLLM. The baseline used two TP=1 instances; prompt-v0 used four TP=1 instances. |
| Qwen3.5-27B | `/workspace/models/Qwen/Qwen3.5-27B`; local checkpoint. Retained runs also record the staged path `/data/anon-user0/old_data/workspace/workspace/models/Qwen/Qwen3.5-27B`. | Local vLLM OpenAI-compatible service at `http://localhost:8000/v1`. The retained model and agent logs do not expose a definitive instance count, TP allocation, or checkpoint revision hash for this run. |

The provider-hosted rows establish only the exact request model name and provider URL observable in the local logs. They do not establish the provider's internal weight version, quantization, serving hardware, or backend revision. The local paths identify the original safetensors directories used by the servers; no additional checkpoint hash was recorded.

## F.2 Sampling, Thinking, Agent Budget, and Retries

The configuration and the emitted request are reported separately because some compatibility routes intentionally omit unsupported fields, while some local wrappers add model-specific fields. "Unset" means the harness did not configure or send the parameter; it does not assert a provider default.

Table 9: Per-family decoding and agent-control settings for baseline and two-stage runs.

| Model / run | Sampling and actual request | Thinking, step limit, and retry policy |
|---|---|---|
| DeepSeek v3.2, GLM 5.1, Claude Opus 4.6 | Configured temperature 0.60; top-p unset. Under the OpenAI-compatible API configuration, temperature was omitted from the emitted payload by `RCA_OMIT_TEMPERATURE=1`. The actual payload used `max_tokens=32768`; the run log's configuration summary records 24,576. | Thinking enabled; `reasoning_effort` empty. Agent step limit 20. Each API request allowed at most five attempts with 5/10/20/40-second exponential backoff; structured extraction allowed three attempts; an incident allowed three task attempts. |
| GPT-5.4 | Configured temperature 0.60; top-p unset; emitted temperature omitted as above. Actual payload `max_tokens=32768`; log summary 24,576. | Thinking enabled with `reasoning_effort=xhigh`. Agent step limit 20. Request, extraction, and incident retry limits were 5, 3, and 3, respectively, with the same request backoff. |
| Gemma 4 12B/31B IT baseline | Configured temperature 0.60, top-p 0.95, and legacy `top_k=20`. Requests used the OpenAI-compatible local service and actually sent `max_tokens=32768`. | Run-level thinking disabled. Agent step limit 20. Request, extraction, and incident retry limits were 5, 3, and 3. |
| Qwen3.5-9B and Qwen3.5-27B local runs | Configured temperature 0.60, top-p 0.95, and legacy `top_k=20`. The Qwen-local request explicitly sent temperature, `repetition_penalty=1.15`, and `enable_thinking=true`; it did not send `max_tokens`, leaving the bound to the vLLM server generation configuration. | The batch-level flag was false, but the Qwen-local emitted payload explicitly enabled thinking; the actual payload is authoritative. Agent step limit 20. Request, extraction, and incident retry limits were 5, 3, and 3. |

For a five-attempt request budget, the four waits occur only after failed non-final attempts; hence the recorded backoff schedule is 5, 10, 20, and 40 seconds. Extraction retries repair malformed structured predictions, whereas task retries restart an incident after execution failure; neither changes the 20-step limit of one agent attempt.

### F.3 Local Hardware, Inference Framework, and Software

All retained local deployments used NVIDIA H100 GPUs. The internal scheduler allocation requested four accelerators; historical task metadata records 80 GB per accelerator. The evidence does not preserve a more detailed board-level SKU string, so we report the user-confirmed H100 model together with the retained 80 GB capacity rather than inventing a more specific part number. TP denotes tensor parallelism; workers are agent-side concurrent jobs assigned to each server instance.

Table 10: Retained local deployment configuration.

| Local model | Hardware and parallelism | vLLM / inference parameters | Software-version evidence |
|---|---|---|---|
| Gemma 4 12B IT | Anonymous scheduler allocation; four H100 80 GB GPUs. Four instances times three workers = 12 concurrent jobs; TP=1. | vLLM 0.18.1, Transformers backend; `gpu-memory-utilization=0.92`, `max-num-seqs=12`, `max-model-len=262144`, `enforce-eager`, and `TRITON_ATTN`. | Python 3.13. Gemma overlay: Transformers 5.16.1, tokenizers 0.23.1, safetensors 0.8.0. Base environment metadata: torch 2.10.0+cu128 and vLLM 0.18.1. |
| Gemma 4 31B IT | Same four-H100 node. Two instances times three workers = 6 concurrent jobs; TP=2. | Same framework and parameters as 12B except `max-num-seqs=6`; `max-model-len=262144`, eager mode, and `TRITON_ATTN` retained. | Same Python, Gemma overlay, torch, CUDA-build metadata, and vLLM versions as the 12B deployment. |
| Qwen3.5-9B baseline | Two H100 GPUs; two TP=1 instances times three workers = 6 concurrent jobs. | vLLM 0.18.1; `gpu-memory-utilization=0.95`, `max-num-seqs=96`, and Triton GDN prefill. `max-model-len` omitted, retaining the native 262,144-token limit. | Python 3.13, torch 2.10.0+cu128, vLLM 0.18.1. |
| Qwen3.5-9B prompt-v0 | Four H100 GPUs; four TP=1 instances times three workers = 12 concurrent jobs. | Same Qwen vLLM parameters as the baseline deployment. | Same Python, torch, CUDA-build metadata, and vLLM versions. |
| Qwen3.5-27B | Local H100 deployment. The retained experiment records identify the checkpoint and localhost vLLM endpoint, but not the exact card count, TP degree, instance count, or server launch flags. | vLLM OpenAI-compatible serving; unrecorded server flags are left unspecified. | The run artifacts do not independently retain a complete package-version manifest for this deployment. |

The software versions above follow the evidence layer in which they were recorded. In particular, Gemma's deployment overlay identifies its Transformers, tokenizers, and safetensors versions, while torch 2.10.0+cu128 and vLLM 0.18.1 come from the base virtual-environment metadata. The "cu128" suffix is build metadata; it is not treated as a separately measured driver or runtime version.

### F.4 Data Availability

The 131 TelcoOutageBench incidents and their 242 expert root-cause annotations are used under a data-use agreement with the telecommunications industry partner that does not permit public redistribution, and they are therefore not released. The evaluation protocol is specified in the main text and appendix: Section 3.1 defines the task and output schema, Section 3.3 specifies the matching-aware evaluator, and Appendix D gives the complete contracts of `filter_alarms`, `restore_paths`, and `diagnose`.

Within the research environment, incident identifiers were replaced with sequential anonymous IDs. Ticket/work-order identifiers, alarm-coding identifiers, free-text internal annotation notes, internal account paths, document metadata, and organization-specific names or domains were removed or replaced with neutral labels. Operational IPv4 addresses were deterministically pseudonymized within `10.0.0.0/8` while preserving address equality and original /24 grouping, so topology joins and tool behavior remain reproducible. This procedure preserves the chronological split, label cardinalities, and the statistics reported in Table 1.