

ASTRA - Agentic System for Ticket Resolution and Analysis

SHASHIDHAR REDDY JAVAJI^{*†}, Stevens Institute of Technology, USA

MOHAMED TRABELSI^{*}, Nokia Bell Labs, USA

JIN CAO, Nokia Bell Labs, USA

HUSEYIN UZUNALIOGLU, Nokia Bell Labs, USA

Technical operations teams resolve large volumes of incidents and support tickets by synthesizing fragmented evidence from ticket text, historical cases, system logs, and technical documentation. This process is slow and error-prone, and existing automation often relies on monolithic, single-shot generation that lacks explicit evidence modeling and provenance, making outputs difficult to verify when critical signals are sparse and scattered across sources. We propose ASTRA, an agentic system for ticket resolution and analysis in which a central orchestrator coordinates three specialist information-gathering agents and drives a judge-orchestrator refinement loop to produce evidence-backed troubleshooting reports. A *TicketSimilarityAgent* retrieves relevant historical precedents through a two-phase dense-retrieval and LLM-reranking pipeline; a *LogAgent* distills hundreds of thousands of log lines into structured, quote-grounded findings via deterministic filtering followed by a constrained LLM analysis; and a *DomainKnowledgeAgent* retrieves relevant documentation and technical knowledge via the Model Context Protocol (MCP). The three agents' outputs are then transformed into an explicit claim-evidence intermediate representation that links every claim to a verbatim source passage, annotates it with a support level, and prevents cross-attribution between agents. A *JudgeAgent* scores the resulting report on five rubric criteria, and an *OrchestratorAgent* translates low-scoring metrics into a small set of targeted follow-up queries; the loop runs for a bounded number of rounds. Evaluated on 987 real-world telecom fault tickets across seven product lines, ASTRA achieves a mean quality score of 4.13/5.0 across five evaluation dimensions, with 59.9% of reports correctly identifying the fault area at the component-family level or better. Presentation quality is consistently high (Relevance 4.88, Clarity 4.94), and the claim-evidence architecture limits fabricated technical details to under 3% of error cases. Stratification by fault type reveals that hardware faults remain substantially harder than software or configuration faults (Cohen's $d=0.80$), pointing to a fundamental limitation of text-based evidence channels for hardware fault diagnosis.

CCS Concepts: • **Computing methodologies** → **Information extraction**; *Lexical semantics*; **Multi-agent planning**; **Multi-agent systems**; **Intelligent agents**; • **Software and its engineering** → *Software maintenance tools*; • **Information systems** → **Language models**; *Similarity measures*.

Additional Key Words and Phrases: ticket troubleshooting, incident management, root cause analysis, on-call automation, log analytics, runbook grounding, retrieval-augmented generation, claim-evidence grounding, tool-augmented language models, multi-agent orchestration, LLM-as-a-judge, provenance and auditability

1 Introduction

Technical operations teams resolve large volumes of incident and support tickets under tight time constraints. A single ticket often requires synthesizing *heterogeneous, partially redundant, and frequently noisy* evidence from (i) the ticket narrative and metadata, (ii) historical tickets and prior resolutions, (iii) system logs / telemetry, and (iv) documentation and troubleshooting guides. In practice, engineers iteratively form hypotheses, gather targeted evidence,

^{*}Contributed equally to this research.

[†]Work done while interning at Nokia Bell Labs.

Authors' Contact Information: Shashidhar Reddy Javaji, sjavaji@stevens.edu, Stevens Institute of Technology, Hoboken, New Jersey, USA; Mohamed Trabelsi, mohamed.trabelsi@nokia-bell-labs.com, Nokia Bell Labs, Murray Hill, New Jersey, USA; Jin Cao, Nokia Bell Labs, Murray Hill, New Jersey, USA; Huseyin Uzunalioglu, Nokia Bell Labs, Murray Hill, New Jersey, USA.

validate (or refute) competing explanations, and finally produce a resolution report that must be both *actionable* and *auditable*—especially in availability-critical domains such as telecommunications [43].

A substantial body of research has targeted *individual components* of the ticket lifecycle. On the “front end”, incident and ticket *triage* systems aim to route or categorize tickets efficiently and consistently; large language models have been shown to provide accurate and interpretable incident triage in production-oriented settings [46], and multi-agent formulations can emulate multi-role workflows during assignment [51]. In parallel, work on *ticket escalation* addresses operational realities where decisions must be topic-aware and dynamic as tickets evolve [30]. On the “back end”, LLM-augmented methods for *ticket aggregation* reduce resolution effort by consolidating related reports and supporting low-cost defect handling [38]. While valuable, these lines of work typically optimize isolated stages rather than supporting evidence-driven, end-to-end troubleshooting.

Recent telecom-focused systems have taken an important step toward integration. TeleDoCTR [43] combines domain-specific ranking with generation to automate routing, similar-ticket retrieval, and fault analysis report drafting outlining the issue, root cause, and potential solutions for telecom troubleshooting. However, end-to-end ticket resolution remains challenging when evidence is distributed across sources with differing reliability and when the system must decide *what to investigate next* to close gaps. More broadly, realistic agent benchmarks suggest that robust IT automation is still far from solved where ITBench reports low end-to-end success rates for state-of-the-art agents on Site Reliability Engineering (SRE)-style scenarios [23], and similarly low end-to-end success persists in realistic web and tool-use environments and software engineering workflows [27, 32, 50, 55]. These observations point to a core challenge beyond *fluency-controllability, verification, and evidence-grounded iteration* in complex, tool-heavy workflows.

In response, the community has increasingly explored *agentic* and *tool-augmented* paradigms for diagnosis and remediation [19, 29, 34, 48]. LLM-based root cause analysis (RCA) systems can automate incident analysis by collecting diagnostic signals and generating explanatory narratives [5], and LLMs can also improve investigation efficiency by recommending domain-specific diagnostic queries grounded in historical incidents [25]. LLex [28] further frames incident handling as the execution of troubleshooting guides by an agent that converts procedural knowledge into actionable plans. At a systems level, STRATUS [4] demonstrates how specialized agents can be organized under explicit control logic and safety constraints for iterative mitigation.

This paper builds on that trajectory and argues that ticket troubleshooting is best approached as orchestrated multi-agent evidence synthesis. We propose an agentic telecom-related ticket troubleshooting framework in which a single orchestrator coordinates specialized agents aligned with key evidence channels: a *TicketSimilarityAgent* retrieves high-utility historical precedents; a *LogAgent* performs structured extraction over large log streams to surface diagnostically “golden” signals; and a *DomainKnowledgeAgent* grounds reasoning in canonical knowledge by querying domain-specific documentation via the Model Context Protocol (MCP). Each agent produces bounded artifacts intended for downstream verification rather than unconstrained narrative text. Because troubleshooting often spans long horizons and repeated interactions, we explicitly account for long-term memory and state management mechanisms inspired by recent agent memory systems [33, 35, 49]. The orchestrator then constructs an explicit *claim–evidence intermediate representation*: candidate claims paired with provenance-linked evidence snippets from one or more sources, annotated with support levels and cross-source consistency signals. This intermediate layer enables controllable report generation, targeted follow-up queries, and auditable traceability.

A key design goal is to avoid the brittleness of monolithic end-to-end generation by using iterative refinement with evidence attribution and explicit planning. RARR [14] shows how generation can be post-edited to add attribution and remove unsupported content, and Self-RAG [2] motivates retrieval-and-critique behaviors [45] that determine when

retrieval is needed and how evidence should shape generation. We operationalize these ideas via a *judge-orchestrator loop*: a judge agent evaluates draft reports against task-specific criteria (e.g., root-cause plausibility and identification quality), flags weak or under-supported sections, and triggers targeted follow-ups to the appropriate specialist agent(s). To reduce unsupported conclusions, we incorporate explicit verification behaviors shown to mitigate hallucinations [9, 31]. Because LLM-based evaluation can be inconsistent, we follow reliability-oriented guidance for “LLM-as-a-Judge” systems [16, 54]. For retrieval-augmented components, we incorporate reference-free diagnostic signals inspired by RAG evaluation frameworks such as RAGAs [11].

In summary, we make the following contributions:

- **An end-to-end multi-agent troubleshooting architecture** that coordinates specialized agents over historical tickets, logs/telemetry, and documentation to produce evidence-backed fault analysis (FA) reports outlining the issue, root cause, and potential solutions.
- **A claim-evidence intermediate representation** with provenance links and cross-source consistency checks to improve traceability and support controlled refinement.
- **A judge-orchestrator refinement policy** with targeted follow-ups and bounded rounds, designed to reduce unsupported conclusions while preserving operational usefulness.
- **An evaluation protocol** that measures report quality alongside evidence grounding and retrieval faithfulness, informed by realistic agent benchmarking [11, 23].

2 Related Work

2.1 Ticket Troubleshooting

Operational troubleshooting has recently been reframed as a sequence of ticket-centric tasks—routing/triage, retrieval of similar historical cases, and generating resolution artifacts. Early LLM-based systems and studies show that language models can support interpretable incident triage decisions [46], while other lines of work improve operator productivity by recommending better diagnostic queries [25, 26]. In parallel, ticket workflow automation has been decomposed into specialized sub-problems such as ticket summarization [42], escalation [30] and aggregation/deduplication for faster defect resolution [38, 39]. Complementary efforts profile and structure ticket content itself (e.g., hierarchical fault profiling) to enable downstream analytics and decision-making [20].

Beyond isolated subtasks, recent systems move toward more end-to-end troubleshooting pipelines. TeleDoCTR [43] explicitly unifies classification, retrieval, and report generation for telecom ticket resolution. In cloud incident settings, LLMs have been used to generate root-cause analyses and mitigation steps [1, 5, 6], and agentic approaches such as RCAgent [47] emphasize tool-augmented interaction for collecting evidence during RCA. Industrial on-call automation has also begun exploring explicit multi-agent collaboration as a design pattern [13]. However, these systems still often rely on brittle orchestration (fixed stages or Standard Operating Procedure (SOP) templates) and do not consistently expose controllable intermediate representations (e.g., explicit hypotheses, evidence sets, and verification outcomes) that operators can audit.

A growing body of work argues that multi-agent decomposition can reduce cognitive load and improve robustness in high-stakes operations. Triangle [51] introduces multi-agent incident triage to route incidents more effectively, and STRATUS [4] explores multi-agent autonomy for reliability engineering in modern clouds. More explicitly RCA-focused multi-agent methods incorporate structured workflows and coordination mechanisms, e.g., SOP-enhanced multi-agent RCA [36], knowledge-base driven troubleshooting workflow coevolution [37], and blockchain-inspired

voting/termination controls for microservices RCA [53]. Even when these approaches improve automation accuracy, they still highlight persistent gaps: (i) *context fragility* when evidence is distributed across tickets, logs, and documentations, (ii) *error propagation* across long multi-step reasoning chains, and (iii) limited *controllability* over what the system asserts vs. what it can justify. Recent multi-agent-RCA work makes these failure modes explicit and motivates specialized agents (retrieval/validation) to counter hallucinations and context-switching errors [12]. These trends collectively motivate a multi-agent fault analysis framework that is not just task-compositional, but also *evidence-grounded and verifiable* by construction.

2.2 Agentic Systems

The broader agentic systems literature provides the scaffolding needed to operationalize such decomposition. Tool augmentation is a foundational capability for agents acting over external state, with TALM [34] demonstrating how non-differentiable tools can be integrated via demonstration and iterative improvement. Planning-centric prompting methods (e.g., Plan-and-Solve) further show that explicit decomposition can reduce missing-step and reasoning errors in multi-step tasks [45]. On the coordination side, multi-agent frameworks range from communication-driven role play [29] and agent conversation abstractions [48] to more structured meta-programming approaches for collaborative software production [19] and reflective collaboration mechanisms that emphasize iterative critique and revision [3].

A recurring challenge in real deployments is that long-horizon tasks stress both memory and environment interaction. Systems such as MemGPT [33] treat memory as a managed hierarchy to extend effective context beyond window limits, while A-Mem [49] proposes agentic memory mechanisms tailored to LLM agents. Meanwhile, realistic evaluation environments and benchmarks expose how difficult it remains to build robust autonomous agents: WebArena [55] provides a grounded web environment for agent interaction, GAIA [32] targets general AI assistants, and software-engineering benchmarks highlight brittleness at the repository level [27] even when agent-computer interfaces are introduced [50]. Multi-agent issue-resolution systems such as MAGIS [40] attempt to address these gaps via specialized roles (manager/developer/QA) but still achieve modest solve rates, underscoring the difficulty of reliable orchestration. Similarly, ITBench [23] shows that state-of-the-art agents resolve only a minority of real-world IT automation scenarios, indicating substantial headroom for reliability-centric agent design.

Taken together, these results suggest that merely adding agents is insufficient to address operational fault analysis demands: (i) stable shared state (memory), (ii) controllable decomposition, and (iii) grounded interaction with tools and evidence. This directly motivates frameworks that explicitly separate *analysis claims* from *supporting evidence* and provide systematic verification hooks, rather than relying on implicit end-to-end generation.

2.3 Claim Evidence

A key missing ingredient in many operational RCA pipelines is a principled claim–evidence interface that supports attribution, verification, and post-hoc auditing [21]. In text generation settings, RARR [14] demonstrates how to retrofit attribution by explicitly researching and revising model outputs. Retrieval-augmented generation methods increasingly incorporate self-critique and reflection (e.g., Self-RAG [2]) to decide when to retrieve, how to critique, and how to revise generations. At the system level, evaluation frameworks such as RAGAs provide reference-free metrics that separately assess retrieval quality and faithfulness of the generated answer [11]. For hallucination reduction, Chain-of-Verification (CoVe) [9] formalizes a staged procedure where models draft, generate verification questions, answer them independently, and then produce a revised response; complementary black-box approaches such as SelfCheckGPT [31] detect inconsistencies by sampling multiple generations.

Reliability also depends on *how* we evaluate complex reasoning outputs. Recent work on LLM-as-a-Judge analyzes judge design, bias, and reliability issues, and proposes methodological standardization for using LLMs as evaluators [16]. Benchmarks like MT-Bench/Chatbot Arena have been used to study judge behavior and its failure modes [54]. In claim verification specifically, surveys document the shift toward LLM- and RAG-based verification pipelines, including common retrieval, prompting, and fine-tuning patterns [10]. Fine-grained, explainable verification systems such as ClaimVer [8] emphasize claim-level verification with evidence attribution and human-centric explanations. Newer datasets push beyond single-hop text-only verification: MMCV [44] targets multi-hop *multimodal* claim verification across text, images, and tables, while CliniFact [52] provides a clinical research claim benchmark that exposes the gap between discriminative and generative verification performance. Finally, system demonstrations like CEDAR [22] highlight cost-efficient, data-driven verification pipelines that are closer to operational constraints.

Overall, the evidence-grounded generation and verification literature indicates that robust systems require explicit intermediate artifacts (claims, evidence, attribution, verification results) plus reliable evaluation. Yet, many ticket/incident systems still produce narrative fault analysis reports without explicit, checkable claim–evidence structure. This motivates a multi-agent fault analysis framework that (i) decomposes troubleshooting into controllable sub-steps, (ii) maintains and curates context over long horizons, and (iii) generates reports as verifiable sets of claims with attached evidence and validation outcomes, aligning operational RCA with best practices in grounded generation and claim verification.

3 Problem

We study the problem of *automated fault analysis report generation* for technical support tickets in telecom infrastructure. Formally, given a target ticket T consisting of structured metadata (product, feature, software build) and unstructured text (title and problem description), along with references to associated log files, the goal is to produce a fault analysis report $R = (\text{identification}, \text{resolution})$ where the *identification* section describes how the fault was detected and what caused it, and the *resolution* section describes the fix that should be applied.

Generating this report requires synthesizing evidence from three heterogeneous external channels alongside the ticket itself: (i) a corpus of historical tickets and their resolved fault analyses, from which relevant precedents can be retrieved; (ii) raw system logs collected from the affected network node, typically comprising hundreds of thousands of log lines drawn from separate system and runtime log streams; and (iii) a body of technical documentation including alarm definitions, configuration guides, and known-issue references. Each channel contributes a different kind of signal—pattern-based precedent, concrete runtime evidence, and normative specification, respectively—and a complete analysis ideally draws on all three.

Several factors make this a non-trivial problem:

Evidence quality and coverage are uneven. Similar historical tickets may describe related but not identical fault scenarios. Log files are large and noisy: critical error indicators are intermixed with routine system events, and the relevant lines may be scattered rather than contiguous. Documentation may cover the component in question but not the specific configuration or version deployed.

Sources must be cross-validated, not just aggregated. A root cause hypothesis that emerges from a historically similar ticket may be inconsistent with what the logs for the current ticket actually show. Conversely, a log-derived signal may be ambiguous without documentation context. Simply concatenating source outputs and feeding them to a generation model risks producing a report that blends conflicting signals without acknowledging the conflict.

Reports must be auditable, not just fluent. In availability-critical environments, a report that *sounds* confident but cannot be verified against the evidence it was derived from is often worse than no report at all. Operators need to

know which parts of an analysis are strongly grounded and which rely on weaker or indirect signals, so that they can triage the report’s conclusions appropriately.

The long-context challenge. Log files attached to tickets can span tens of thousands of lines. Passing the full log stream as LLM context is expensive and tends to dilute the model’s attention away from the relatively small number of lines that are diagnostically informative. A principled selection strategy is needed that is both accurate (does not discard critical lines) and conservative (does not hallucinate evidence that is not present).

We further restrict the generation target to a *structured, evidence-backed* report in which every substantive claim is linked to the source that supports it, and claims are explicitly annotated with a support level so that downstream evaluation can distinguish well-grounded findings from weaker inferences. This framing enables our system to generate outputs that are both practically useful for engineers and amenable to automated evaluation.

4 ASTRA

We describe ASTRA (**A**gentic **S**ystem for **T**icket **R**esolution and **A**nalysis), a multi-agent framework for automated fault analysis report generation. ASTRA coordinates three specialist information-gathering agents, a structured claim-evidence construction pipeline, and a judge-orchestrator refinement loop (Figure 1).

The central design decision in ASTRA is to separate *information gathering* from *report generation*, with an explicit intermediate representation sitting between them. Rather than directly prompting a model to produce a fault analysis from raw ticket context, ASTRA first collects and structures evidence through specialized agents, constructs a claim-evidence mapping that annotates each finding with its source and confidence level, and then generates the report from this structured representation. This decoupling makes intermediate outputs inspectable, reduces the surface area for hallucination, and allows the refinement loop to identify and address specific gaps rather than revising the report holistically.

4.1 Agent Architecture

ASTRA operates three specialist information-gathering agents, each aligned with one of the external evidence channels described in Section 3. The agents are implemented on top of AutoGen’s agent communication framework [48], and each exposes a uniform `run(task)` interface so the orchestrator can query them identically during initial evidence collection and in follow-up rounds.

TicketSimilarityAgent. This agent retrieves historical tickets that are semantically similar to the target and returns formatted summaries of their fault analyses and resolutions as its output artifact. Internally it uses a two-phase retrieval strategy (full reranker prompts in Appendix A.1). In the first phase, dense vector search over a database of approximately 130,000 enhanced historical tickets fetches a broad candidate set of up to 80 tickets. Our dense retriever is an MPNet-initialized domain-specific ranker trained with large-batch contrastive learning (in-batch negatives; GradCache [15] for memory efficiency), which yields substantially stronger telecom retrieval than vanilla MPNet. Ticket representations in this database have been enriched with LLM-generated expanded descriptions, acronym expansions, and component context during an offline preprocessing step, which improves query coverage for abbreviated or domain-specific fault descriptions. In the second phase, an LLM-based reranker proceeds in two passes. The first pass rescores the 80 candidates in four batches of 20 and selects the top-5 from each batch (20 intermediate results); the second pass rescores those 20 and selects the final top- k (we use $k = 5$), discarding any candidate whose reranker-adjusted similarity score falls below 0.65. For the top two ranked results, the full investigation trail recorded by the technical analysis (TA) team (where available) is appended to the ticket representation, giving downstream stages richer context about how

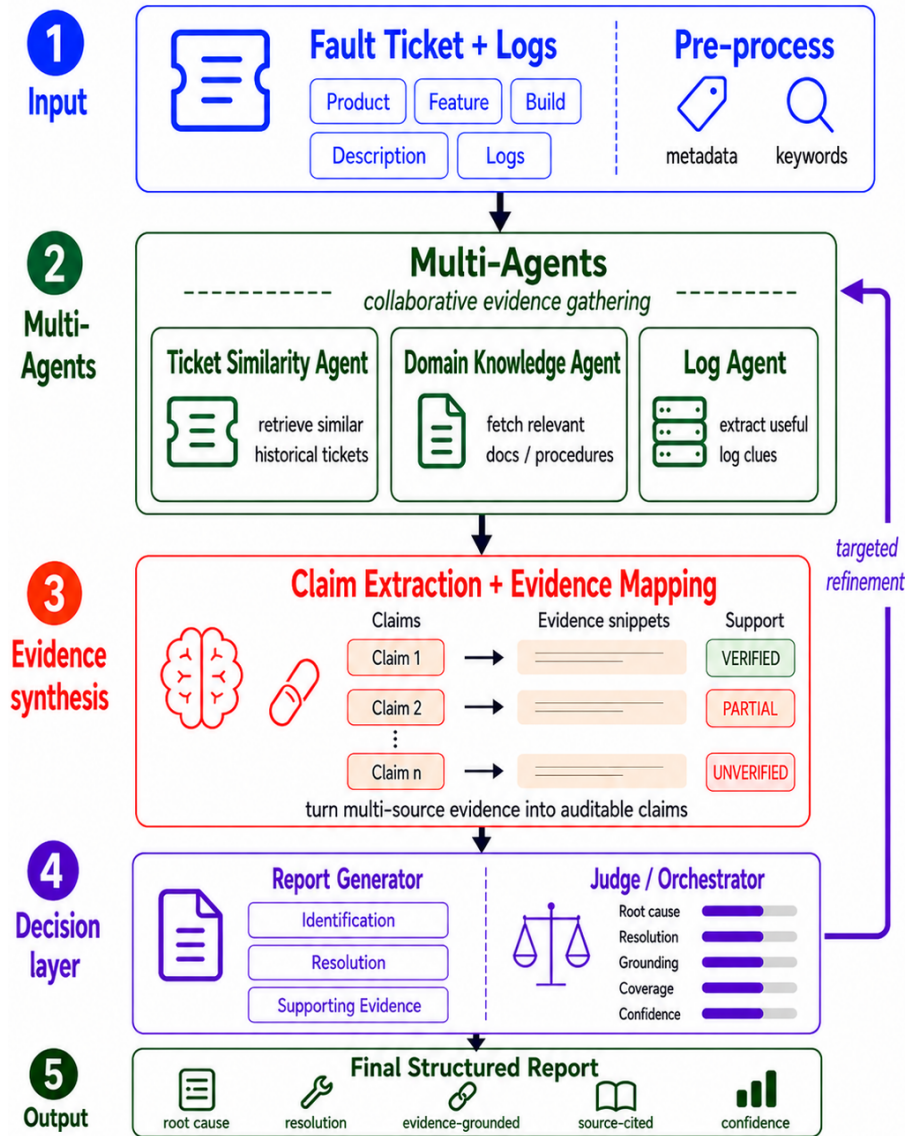


Fig. 1. ASTRA overview. Starting from a fault ticket and preserved logs, ASTRA invokes three specialized agents (TicketSimilarityAgent, DomainKnowledgeAgent, and LogAgent) to gather complementary evidence. Their outputs are converted into a claim-evidence intermediate representation with support labels, which is then used to generate a structured report. A Judge/Orchestrator scores each draft against rubric criteria and issues targeted follow-up queries to the most relevant agents for a bounded number of refinement rounds before producing the final report.

engineers previously diagnosed similar faults. As described in the execution schedule below, this two-phase pipeline is run once per ticket as a shared pre-fetch whose output is consumed both by this agent and by the LogAgent.

LogAgent. Software logs play a central role in diagnosis and automation tasks, as they often constitute the primary source of information capturing software runtime behavior. These collected logs are leveraged in a wide range of log

mining applications, including anomaly detection [17, 41], failure prediction [7], and failure diagnosis [24]. LogAgent analyzes the raw system and runtime log files attached to a ticket and returns a structured analysis containing diagnostic findings, a causal chain, and a root cause category. ASTRA processes log files in full without truncation, with individual log streams that can contain hundreds of thousands of lines. The pipeline proceeds in four stages:

Loading: both syslog and runtime log parquet files are read into memory in full, with severity labels derived from the feature name column (ERR, WRN, CRIT, etc.).

Deterministic filtering: a five-level filter identifies diagnostically relevant lines without any LLM involvement. Level 1 selects lines whose severity label indicates an error or warning. Level 2 matches a curated set of error and fault keywords in the message text (alarm, fault, fail, exception, crash, timeout, etc.). Level 3 captures state-transition events that are diagnostically meaningful even at informational severity (restart, deregister, handover failure, sync loss, etc.). Level 4 targets component-level anomaly indicators (core dump, stack trace, watchdog, queue overflow, etc.). Level 5 extracts keywords from the ticket’s own title and description *and* from the titles and descriptions of the pre-fetched similar tickets, using them to catch relevant lines specific to the reported component or feature that generic patterns would miss. Results from all five levels are unioned and deduplicated by line index. A noise suppressor then identifies high-volume background components that account for more than 25% of diagnostic lines but contain no fault-severity keywords (e.g., transport-layer protocol chatter) and replaces their individual lines with a single metadata annotation so they do not anchor the LLM’s attention.

Context expansion: each retained diagnostic line is expanded to include ± 5 surrounding lines from the original log stream, providing the LLM the narrative flow around each event. If the combined set exceeds an upper limit, DRAIN3 [18] template clustering is applied to deduplicate structurally similar lower-severity lines while preserving all distinct error and warning messages. The post-filtering excerpt is capped at 2,000 lines before being passed to the LLM; 84% of tickets in our evaluation saturate this limit.

Constrained LLM analysis: the resulting log excerpt is passed to the LLM together with the ticket’s enhanced description, acronym expansions, and component context, and—crucially—the shared pre-fetch data drives two distinct behaviours here. First, as noted above, keywords from similar ticket titles and descriptions augment Level 5 filtering to surface log lines specific to the fault’s component. Second, the resolution summaries and root cause labels from the pre-fetched tickets are formatted with rank labels (HIGH MATCH for ≥ 1.0 reranker score, GOOD MATCH for ≥ 0.9 , and POSSIBLE MATCH for scores below 0.9) and appended to the LLM prompt as a labelled reference block (“SIMILAR RESOLVED HISTORICAL TICKETS”), with TA investigation trails included for the top two ranked matches only—TA trails can extend to several hundred lines of debugging notes, so including them for all five candidates would inflate the prompt without proportional diagnostic benefit, given the lower transferability of lower-ranked resolutions; the LLM is instructed to use these only to cross-validate its log findings—not to copy them verbatim (Appendix A.2). The output is a structured JSON object with a causal narrative, the supporting log evidence, and a root cause category selected from a fixed taxonomy of 53 categories derived from the historical ticket database.

DomainKnowledgeAgent. This agent retrieves content from technical documentation and returns findings relevant to fault identification and resolution (full prompt in Appendix A.3). It queries a domain knowledge base via the MCP, which provides access to alarm specifications, configuration guides, and known-issue references.

The query issued to the MCP backend is constructed to be as targeted as possible. It incorporates the ticket title, any fault identifier codes (FIDs) extracted from the title via regex, and—crucially—up to 1,500 characters of TicketSimilarityAgent output and 2,000 characters of LogAgent output. This means the DomainKnowledgeAgent can issue queries like “what configuration parameter causes this alarm given that the log shows X” rather than a generic alarm

lookup. The agent’s system message instructs it to distinguish identification-relevant findings (symptom meanings, diagnostic procedures, alarm definitions) from resolution-relevant findings (configuration changes, workarounds, known-issue fixes) in its output, which aligns with the two-section structure of the fault analysis report.

Execution schedule. Execution proceeds in three steps. First, a single synchronous pre-fetch runs the two-phase retrieval pipeline and stores the results as raw structured dictionaries—one per retrieved ticket—carrying root cause labels, resolution text, reranker scores, and TA investigation trails for the top two results. These dictionaries are shared between both Phase 2a agents: the TicketSimilarityAgent formats them into the human-readable summaries that constitute its output artifact, while the LogAgent receives them at construction time to drive Level 5 keyword filtering and the reference block injected into its LLM prompt. Because both agents consume the same pre-fetched data, they are mutually input-independent at the start of Phase 2a and execute concurrently via `asyncio.gather`. Second, once Phase 2a completes, the DomainKnowledgeAgent runs in Phase 2b; the orchestrator embeds up to 1,500 characters of TicketSimilarityAgent output and 2,000 characters of LogAgent output directly into its task string, allowing it to issue documentation queries conditioned on both agents’ findings. Third, the three agent outputs are passed to the claim–evidence construction pipeline before final report generation and refinement.

4.2 Claim–Evidence Intermediate Representation

Once the three agents have produced their outputs, ASTRA constructs a structured intermediate representation before generating the fault analysis report. This proceeds in two steps.

Claim extraction. Each agent’s output is processed independently to extract a set of detailed, self-contained diagnostic claims; the extraction prompt is shown below. The extraction model is instructed to produce 5–10 claims per agent, annotating each with a [CAUSAL] prefix where the claim describes a root cause or direct trigger, and a [SYMPTOMATIC] prefix where it describes an observed downstream effect. When the distinction is unclear, no label is applied as we found that forcing uncertain labeling introduced more noise than leaving claims unlabeled.

A key constraint is that claims must be grounded strictly in the agent’s own output. Agent-specific system messages explicitly prohibit cross-attribution: TicketSimilarityAgent claims must describe what retrieved historical tickets show (citing their problem report identifiers and similarity scores), LogAgent claims must reference only entries visible in the log analysis output, and DomainKnowledgeAgent claims must cite only content appearing in the retrieved documents. This discipline is important in practice because the same log line or historical resolution could appear plausibly relevant to all three agents, and without explicit attribution boundaries the model would tend to repeat the same finding in all three claim sets with slightly different framing, producing a misleadingly inflated pool of evidence.

Claim Extraction Prompt

Extract detailed, information-rich claims for a telecom fault analysis report.

Focus on two sections:

1. IDENTIFICATION -- How was the problem detected and diagnosed?

Include: alarm IDs, log messages, error patterns, component names, root cause.

2. RESOLUTION -- How was the problem resolved?

Include: exact fixes, configuration parameters (before/after), commands, workarounds, and verification steps.

CRITICAL GROUNDING RULE:

- Extract ONLY information explicitly stated in the AGENT OUTPUT provided.
- The target ticket context is background only -- do NOT use it to invent claims.
- Do NOT cross-attribute between agents.
- When in doubt, omit.

```
CAUSAL vs SYMPTOMATIC LABELLING (use where unambiguous):
[CAUSAL]    -- direct trigger or root cause
[SYMPTOMATIC] -- downstream effect or observed symptom
(omit label if unsure)
CLAIM QUALITY:
- Each claim must be self-contained with specific technical details verbatim from output.
- Broader observational findings are valuable even without a specific ID.
- Negative findings (e.g., "no further alarms after fix") should be included.

Output: one claim per line. Extract 5-10 claims covering identification and resolution.
```

Evidence mapping. For each extracted claim, ASTRA uses the LLM to locate the specific passage in the originating agent output that most directly supports the claim (shown below). The extraction prompt instructs the model to quote the text verbatim, prefer passages containing exact identifiers (error codes, module names, parameter values) over paraphrases, and include one or two surrounding sentences for context where they add diagnostic value. The extracted evidence is validated by checking that at least 60% of its words appear in the original agent output; this is a sourcing fidelity check—it detects fabricated technical identifiers (error codes, parameter names, module paths) that would score highly on semantic similarity but do not actually appear in the source—rather than a semantic retrieval step. Passages falling below this threshold are flagged as potentially hallucinated and replaced by a keyword-based fallback that scores sentence windows by keyword overlap. Each evidence item is annotated with a confidence level (high, medium, or low) and a status (supported or weak), and items meeting the confidence threshold are tagged as [VERIFIED] for the downstream generation step. Items that do not pass are retained but marked [UNVERIFIED] so the report generator can handle them conservatively.

```
Evidence Mapping Prompt
-----
You are a precise evidence extraction assistant for telecom fault reports.
Find the EXACT text from the agent output that supports each claim.

CRITICAL RULES:
1. QUOTE DIRECTLY -- copy exact text, do NOT paraphrase or summarize.
2. MOST DIAGNOSTIC LINE FIRST -- prefer passages with error codes, fault IDs, or parameter values; do not skip
   observational evidence (e.g., load patterns, timing, state sequences) just because they lack a specific ID.
3. SURROUNDING CONTEXT -- include 1-2 sentences before/after if they add value.
4. NO INVENTION -- if no supporting text exists, return "No direct evidence found" and mark status as weak.

Return JSON:
{
  "claim":    "<original claim>",
  "evidence": "<exact quoted text from agent output>",
  "source":   "<agent name>",
  "status":   "supported" | "weak",
  "missing":  "<if weak, what specific detail is absent>",
  "confidence": "high" | "medium" | "low"
}

Confidence:
high  -- evidence directly and specifically supports the claim
medium -- evidence is in the right area but lacks full specificity
low   -- connection requires significant inference
```

4.3 Report Generation and Refinement

Initial report generation. The fault analysis report is generated from the claim–evidence mapping using a structured prompt that enforces a causal narrative format (see Appendix A.4). The prompt specifies that the *identification* section must trace an event chain from the initial symptom through the diagnostic evidence to a confirmed or likely root cause, and the *resolution* section must name the specific fix applied along with verification that the issue was resolved. Claims labeled [VERIFIED] are treated as factual; [UNVERIFIED] claims may be included but must be flagged with hedged language. If the TicketSimilarityAgent signals a potential root cause category from historical matches, this is injected as a low-weight background hint with explicit instructions not to use it to override log evidence.

Judge evaluation. After the initial report is generated, a JudgeAgent evaluates it against five criteria (see Appendix A.5). The judge receives the ticket context, all three agent outputs, the claim–evidence mapping, and the draft report; it has no access to the ground-truth FA or TA investigation chain, scoring purely on internal consistency and evidence grounding. The five criteria are: *Root Cause Identification* (does it name a specific mechanism and component?), *Resolution Specificity* (are the fix steps concrete and grounded?), *Evidence Grounding* (what fraction of specific technical identifiers in the report can be traced back to agent outputs?), *Coverage and Completeness* (are all four expected dimensions—symptom, root cause, fix, and verification—present?), and *Focus and Relevance* (does the report stay on topic or build its narrative around off-topic agent findings?). Each criterion is scored 1–5, and a score at or below 3 on any criterion triggers a revision. The threshold is intentionally set at 3 rather than the usual midpoint of 2 because based on our experiments with multiple tickets, a score of 3 in our rubric corresponds to a report that addresses the right topics but lacks grounding specificity—a condition that additional targeted queries can often remedy.

Orchestrator planning and refinement loop. When revision is needed, an OrchestratorAgent receives the judge’s evaluation and plans a small set of targeted follow-up queries (typically two to four) directed at the agent or agents most likely to fill the identified gaps. For example, if the root cause identification scored 3 because the log analysis did not pinpoint the triggering component, the orchestrator issues a focused LogAgent query asking specifically about that component. Query deduplication is maintained across rounds so that the same query is never re-submitted in a subsequent iteration. The follow-up results are appended to the original evidence, and the report is revised against both the original claim–evidence mapping and the new findings. This loop runs for up to five rounds; per-round metric changes and convergence behavior are reported in Section 5.7.

Orchestrator Planning Prompt

You are an Orchestrator Agent coordinating information gathering to improve a Fault Analysis Report. You receive a judge evaluation and decide which helper agents to call and what to ask them.

TASK: Create a focused plan to gather only the information that will directly address metrics scored ≤ 3 .

DECISION FRAMEWORK:

Root Cause ≤ 3 : Need diagnostic specificity.

- > LogAgent for exact component/error pattern in logs.
- > DomainKnowledgeAgent for component behaviour specs.
- > TicketSimilarityAgent for historical RCA confirmation.

Resolution ≤ 3 : Need concrete fix details.

- > DomainKnowledgeAgent for exact parameter names, commands, or configuration values.
- > TicketSimilarityAgent for specific steps from a proven resolution.

Coverage ≤ 3 : Identify missing dimension (symptom/root cause/fix/preventive).

- > Query the most relevant agent for that dimension.

Focus ≤ 3 : FA distracted by off-topic evidence.

- > Re-query with questions specific to the fault's component and symptom.

CONSTRAINTS:

```

- 2-4 queries total; more queries add noise without proportional benefit.
- Avoid asking multiple agents for the same information.
- Do not re-submit a question already asked in a prior round.
Return JSON:
{ "revision_needed": true | false,
  "reasoning": "brief explanation of the query plan",
  "agent_queries": [
    { "agent_name": "...", "query": "...", "purpose": "which metric this addresses" }],
  "expected_outcome": "what these queries should add to the report"}

```

5 Evaluation

5.1 Dataset and Setup

We evaluate ASTRA on 987 real-world telecom fault tickets drawn from a large internal corpus that spans seven product lines covering radio access, baseband, and integrated cell-site equipment. Each ticket carries structured metadata (product, feature, software build), an engineer-written description, and references to system-log files. Ground-truth fault analyses (identification + resolution) are written by domain engineers following a standardized FA template.

We additionally have access to a *Technical Analysis* (TA) investigation chain: multi-round debugging notes written by domain engineers during live diagnosis, containing raw log snippets, component names, parameter values, and final conclusions. The TA data is used only by the evaluation judge—ASTRA never sees it at inference time—providing a richer reference against which to measure diagnostic depth.

Evaluation protocol. Each ASTRA-generated report is scored by an LLM judge (GPT-OSS 120B at temperature 0) on five dimensions using a 0–5 Likert scale: *Accuracy*, *Completeness*, *Relevance*, *Clarity*, and *TechnicalDepth* (full evaluation prompt in Appendix A.6). The first four are assessed against the ground-truth FA alone; *TechnicalDepth* is additionally scored using the TA investigation chain as reference, capturing whether the system provides grounded technical details consistent with the engineers’ debugging record. The five scores are averaged into a composite *Overall* score. Scores ≥ 3 indicate a report that is *useful in practice*: the fault area is correctly identified even if the exact mechanism differs from the ground-truth. The Accuracy rubric uses explicit per-score anchors to reduce scoring ambiguity: score 5 requires the same specific root cause mechanism as the ground-truth; score 3 requires the correct fault area and symptom but allows a different specific mechanism; score 1 requires a wrong fault domain entirely. This hierarchical design follows calibration recommendations for LLM-as-a-Judge systems [16, 54].

Table 1. Evaluation results across 987 fault analysis reports (final refinement round, 0–5 scale). *Overall* is the 5-metric average.

Metric	Mean $\pm$ Std	Med.	≥ 3 (%)
Accuracy	2.63 $\pm$ 0.90	3	59.9
Completeness	4.37 $\pm$ 0.70	4	98.4
Relevance	4.88 $\pm$ 0.59	5	97.9
Clarity	4.94 $\pm$ 0.24	5	100.0
Tech. Depth	3.83 $\pm$ 0.85	4	92.0
Overall	4.13$\pm$0.43	4.2	98.4

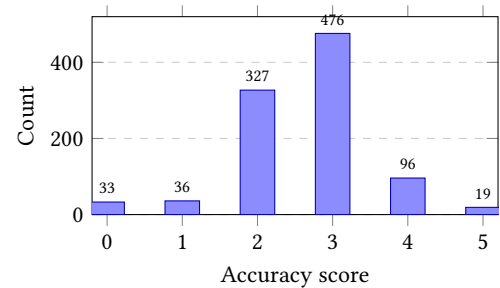


Fig. 2. Distribution of Accuracy scores ($n=987$, final refinement round). The distribution is unimodal and left-skewed (skewness= -0.44), with the mode at 3.

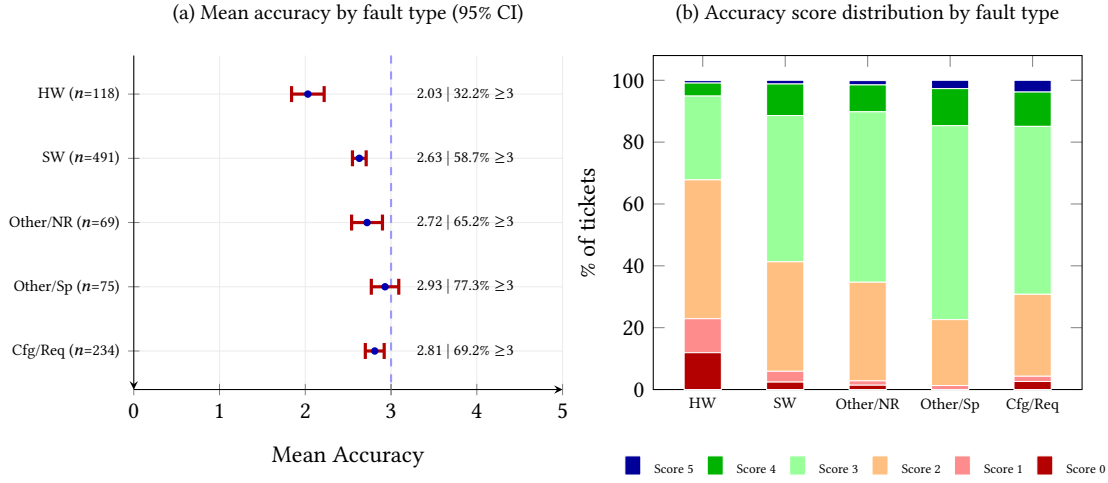


Fig. 3. Accuracy stratified by fault type. (a) Horizontal dot plot with 95% confidence intervals ; the dashed line marks the “useful” threshold (score 3). Hardware faults have the lowest mean (2.03) and the widest CI, with only 32.2% scoring ≥ 3 . (b) Stacked bars show the proportion of tickets in each accuracy bucket. Nearly 68% of hardware tickets score ≤ 2 (red–orange), compared to $\sim 31\%$ for Config/Requirement and $\sim 41\%$ for Software; hardware has almost no score-5 tickets.

5.2 Overall Performance

Table 1 reveals a clear two-tier structure. *Presentation quality* is consistently high: Relevance (4.88), Clarity (4.94), and Completeness (4.37) show ceiling effects with 97–100% of reports exceeding the “useful” threshold. TechnicalDepth (3.83) is strong but more variable (92.0% ≥ 3): as the refinement loop broadens contextual coverage, it sometimes replaces low-level specific details—error codes, parameter values, log identifiers—with more general explanatory content (see Section 5.7). This confirms that the claim–evidence architecture reliably produces well-structured, coherent reports. *Accuracy* (2.63) remains the primary bottleneck, indicating that pinpointing the exact causal mechanism—rather than the general fault direction—is the primary challenge. The accuracy distribution (Figure 2) concentrates at score 3 (48.2% of reports), meaning the system typically identifies the correct symptom and fault area but proposes a different specific mechanism than the ground-truth. Scores 4–5 (exact or near-exact root cause) account for 11.7%, while catastrophic failures (scores 0–1) remain rare at 7.0%.

5.3 Stratification by Fault Type

We classify tickets into five fault-type categories based on the ground-truth root cause label and examine where ASTRA succeeds and where it struggles. Figure 3a reveals that Hardware is the only fault type whose median falls below the “useful” threshold: 67.8% of HW tickets score ≤ 2 , compared to 41.3% for Software and 30.8% for Config/Requirement. The effect is large (Cohen’s $d=0.80$ between HW and Cfg/Req; Kruskal–Wallis $H=64.2$, $p<0.001$ across all five fault types). Figure 3b makes the distributional difference vivid: the Hardware bar is dominated by red and orange (scores 0–2), while the other categories are dominated by green (score 3). Other/Specialized and Config/Requirement faults achieve the highest proportion of scores 4–5 (~ 14 – 15%), consistent with these faults being explicitly documented in ticket narratives. The gap is specifically in diagnosing the hardware-level root cause, which requires board-level telemetry and hardware test data that ASTRA cannot access.

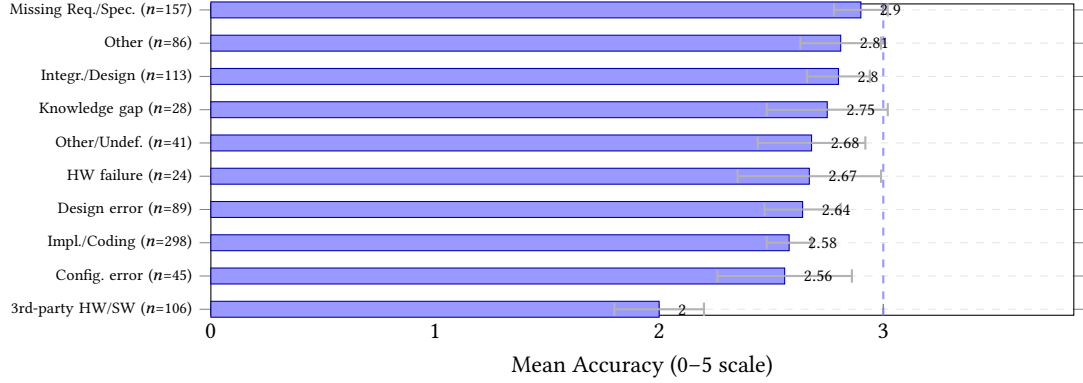


Fig. 4. Mean Accuracy by ground-truth root cause category, sorted by mean accuracy (final refinement round, 95% CI error bars). The dashed line marks the “useful” threshold (score 3). Faults attributable to 3rd-party hardware/software are hardest (Cohen’s $d = -0.74$ vs. all other categories, $p < 0.001$); missing requirement/specification faults are most diagnosable, as their causal evidence appears directly in ticket narratives and documentation. Kruskal–Wallis $H = 65.1$, $p < 0.001$.

5.4 Root cause category analysis.

To complement the coarse fault-type view, we further stratify by the ground-truth *root cause category* recorded in each ticket (Figure 4). Ten categories emerge after grouping the 30+ raw labels. Kruskal–Wallis across categories is highly significant ($H = 65.1$, $p < 0.001$), confirming that root cause type is a strong predictor of diagnosability.

Two poles stand out. *3rd-party HW/SW* is by far the hardest category: mean accuracy 2.00, only 33.0% ≥ 3 , Cohen’s $d = -0.74$ vs. all other categories ($p < 0.001$). These tickets—103 of the 106 are Hardware-typed—involve root causes outside the operator’s own codebase, so neither historical ticket precedents, nor log patterns, nor documentation provide reliable diagnostic signals; the system can identify the symptom domain but cannot pinpoint the third-party source. Conversely, *Missing Req./Spec.* tickets are the most diagnosable (Acc=2.90, 72.0% ≥ 3): these faults stem from explicit gaps in documented specifications, which are directly captured in ticket narratives and documentation—exactly the evidence channels ASTRA is designed to exploit. Integration/design faults (Acc=2.80) follow a similar logic: the fault manifests as a cross-component inconsistency that is often traceable through log sequences and similar historical tickets. This consistency confirms that the difficulty gradient is structural—driven by information availability in the evidence channels—rather than a scoring artifact.

5.5 Error Analysis

To characterize failure modes, we analyze the 396 reports with Accuracy ≤ 2 (40.1% of all reports). Categories are derived from accuracy score levels and qualitative inspection are non-exclusive; counts in Table 2 reflect the relative frequency of each failure pattern. The dominant failure mode (Table 2) is a *plausible but incorrect component attribution* (271 tickets, 68.4%). In these cases the system identifies a related component in the correct product domain—consistent with correct fault-area identification—but not the true responsible unit. The evidence channels surface enough signal to localize the general fault area but not enough to pinpoint the specific component or mechanism, a structurally hard problem given the information asymmetry between ASTRA’s evidence (logs, similar tickets, documentation) and the live debugging sessions available to engineers.

Table 2. Error taxonomy for low-accuracy predictions (Accuracy ≤ 2 , $n=396$, 40.1% of reports). Categories are non-exclusive.

Failure Mode	Count	%
Plausible but incorrect component	271	68.4
Wrong component or subsystem	108	27.3
Wrong fault domain (Acc ≤ 1)	69	17.4
Generic/boilerplate analysis	22	5.5
Fabricated technical details	<10	<3.0

Table 3. Mean evaluation scores by refinement round (paired analysis, $N=674$ tickets that completed all five rounds).

Round	N	Acc	Comp	Rel	Clar	TD	Overall
R0	674	2.72	4.26	4.94	4.91	4.36	4.24
R1	674	2.65	4.34	4.90	4.93	3.86	4.14
R2	674	2.64	4.37	4.90	4.94	3.91	4.15
R3	674	2.62	4.36	4.86	4.94	3.90	4.14
R4	674	2.63	4.36	4.84	4.93	3.83	4.12

Completely wrong component attribution affects 27.3% of error cases, and 17.4% involve the fault being mis-classified at the domain level (Accuracy ≤ 1). Generic or boilerplate analysis, where the report could apply to any ticket of the same class, accounts for 5.5%. Critically, *fabricated technical details remain rare* (<3% of low-accuracy cases), suggesting that the claim–evidence architecture effectively constrains confabulation even when the diagnostic conclusion is incorrect.

5.6 What Drives Accuracy

We compute the Pearson correlation of each metric with Accuracy ($n=987$). Among the five evaluation metrics, Completeness shows the strongest association ($r=0.32$), followed by Relevance ($r=0.28$), Clarity ($r=0.22$), and TechnicalDepth ($r=0.15$). All four are weak to moderate, confirming that the evaluation dimensions capture largely independent quality aspects rather than redundantly measuring a single construct. This separation also explains why the Overall score remains high even when Accuracy is modest—the presentation metrics, which consistently approach their ceiling, dominate the composite. **Report length is not a proxy for quality.** Longer final reports are *not* more accurate. Dividing reports into quintiles by character count shows a slight negative trend in mean Accuracy (Q1: 2.75, Q5: 2.59), indicating that the system does not compensate for diagnostic uncertainty by producing more text. This is an important sanity check on the refinement loop: the 19% length growth across rounds reflects the accumulation of new evidence, not stylistic padding, and accuracy gains (where they occur) come from evidence specificity rather than verbosity.

5.7 Refinement Loop Effectiveness

ASTRA’s judge–orchestrator loop runs for up to five rounds per ticket. To enable a clean paired comparison, Table 3 reports mean scores across the 674 tickets that completed all five rounds, so the same ticket population is tracked from R0 through R4. An important interpretive note: the pipeline judge’s criteria measure *structural completeness*—whether the report covers all expected dimensions with grounded identifiers—not diagnostic correctness. The loop is therefore best understood as a coverage-refinement mechanism, not an accuracy-improvement mechanism. The most striking effect of refinement is on TechnicalDepth. The paired R0→R4 mean delta is $\Delta = -0.53$ ($p < 0.001$), with 47.2% of tickets losing technical depth and only 10.5% gaining. This reflects a systematic side effect: as the judge directs the orchestrator toward

broader contextual elaboration to fill coverage gaps, the LLM tends to replace low-level specific details—error codes, parameter values, log-identified identifiers—with more general explanatory content. Completeness modestly improves ($\Delta = +0.10$, 27.4% improve, 21.5% degrade, $p < 0.001$), confirming that the loop achieves its structural objective—broader coverage—but at a cost to technical grounding. Accuracy is largely unchanged ($\Delta = -0.09$, 16.2% improve, 24.8% degrade, 59.1% unchanged, $p < 0.001$), which is the expected outcome: once the available evidence is exhausted, further iterations can only rephrase and expand; they cannot supply missing diagnostic signal. Relevance ($\Delta = -0.10$) and Clarity ($\Delta = +0.02$) remain near-ceiling throughout, confirming that the refinement loop does not disturb these scores. Of the 987 tickets, 6.8% converge at round 0 (the judge endorses the initial draft immediately), while 68.3% exhaust all rounds. This reveals a natural difficulty gradient: tickets with strong corroborating signals across evidence channels converge quickly; diagnostically ambiguous tickets iterate without necessarily achieving higher accuracy.

6 Conclusion

We proposed ASTRA, a multi-agent system for automated fault analysis that coordinates specialized agents over heterogeneous evidence channels—historical tickets, system logs, and technical documentation—through a structured claim–evidence intermediate representation and an iterative judge–orchestrator refinement loop. Evaluated on 987 real-world telecom fault tickets, ASTRA achieves a mean quality score of 4.13/5.0 across five evaluation dimensions, with 59.9% of reports correctly identifying the fault area. Presentation quality is consistently high (Relevance 4.88, Clarity 4.94), and the claim–evidence architecture limits fabricated technical details to under 3% of error cases. Stratification by fault type reveals that configuration and software faults are substantially more amenable to automated analysis than hardware faults ($d=0.80$), pointing to a fundamental limitation of text-based evidence channels for hardware fault diagnosis. The dominant failure mode—plausible but incorrect component attribution (68.4% of Accuracy ≤ 2 cases)—highlights an inherent ceiling: resolving root cause ambiguity often requires the kind of live debugging and internal knowledge transfer that ASTRA’s evidence channels cannot yet provide.

7 Limitations

Several limitations should be noted. First, evaluation relies on a single LLM judge (GPT-OSS 120B); despite explicit per-score anchors and zero temperature, judge scores may not perfectly align with expert human judgment. Validating against engineer ratings is an important next step. Second, the 2,000-line cap on diagnostic log extraction constrains the LogAgent; 84% of tickets saturate this cap, which may limit the system’s ability to surface rare diagnostic signals buried deeper in the logs. Third, ASTRA has no access to hardware-level diagnostic data—board sensor readings, hardware event logs, or physical component test results—which limits effectiveness on hardware faults, the category with the lowest accuracy (2.03). Fourth, the evaluation covers a single organization’s ticket corpus; generalization to other operators or domains requires further validation.

8 Future Work

Several directions emerge from this work. *Adaptive log budgets*: dynamically adjusting the log extraction cap based on ticket complexity and initial signal strength could improve accuracy for the 68.3% of tickets that currently exhaust all refinement rounds. *Refinement-aware training*: the refinement loop currently improves coverage (Completeness +0.10) but degrades technical specificity (TechnicalDepth −0.54); training the report generator to preserve evidence-grounded details across iterations could recover this trade-off. *Hardware-aware agents*: integrating hardware-level diagnostic data (board sensor readings, hardware event logs, component test results) as an additional evidence channel could address

the domain-level misclassifications that affect 17.4% of low-accuracy cases. *Human-in-the-loop evaluation*: conducting a controlled study with domain engineers to validate LLM-based scoring against expert judgment and measure time savings when engineers use ASTRA reports as starting points. *Retrieval-aware training*: fine-tuning the ticket similarity model to optimize for root cause category overlap, rather than surface-level symptom similarity, could reduce the dominant “plausible but incorrect component” failure mode.

Acknowledgments

We thank Gabriel Górski, Jakub Kozerski, Bartłomiej Ruszaj, and Adrian Dudycz from Nokia Mobile Infrastructure for collecting the telecom-related troubleshooting tickets, fault analyses, technical analyses, and logs. We thank Johann Daigremont from Nokia Bell Labs for creating the MCP server used by the DomainKnowledgeAgent. We thank Ahmet Akyamac from Nokia Bell Labs for collecting the internal acronyms used to augment tickets during the offline preprocessing step before executing the agentic workflow. We thank our past summer intern Yuting Hu for implementing the first version of the agentic workflow using Nokia’s ticket troubleshooting data.

References

- [1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, Melbourne, Australia, 1737–1749. doi:10.1109/ICSE48619.2023.00149
- [2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=hSyW5go0v8>
- [3] Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. 2024. Reflective Multi-Agent Collaboration based on Large Language Models. *Advances in Neural Information Processing Systems* 37 (Dec. 2024), 138595–138631. doi:10.52202/079017-4397
- [4] Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan R. Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. 2025. STRATUS: A Multi-agent System for Autonomous Reliability Engineering of Modern Clouds. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=fYW1PKawwJ>
- [5] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 674–688. doi:10.1145/3627703.3629553
- [6] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. ACM, Athens Greece, 674–688. doi:10.1145/3627703.3629553
- [7] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *The World Wide Web Conference, WWW 2019*. ACM, 2659–2665.
- [8] Preetam Prabhu Srikanth Dammu, Himanshu Naidu, Mouly Dewan, YoungMin Kim, Tanya Roosta, Aman Chadha, and Chirag Shah. 2024. ClaimVer: Explainable Claim-Level Verification and Evidence Attribution of Text Through Knowledge Graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 13613–13627. doi:10.18653/v1/2024.findings-emnlp.795
- [9] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-Verification Reduces Hallucination in Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 3563–3578. doi:10.18653/v1/2024.findings-acl.212
- [10] Alphaeus Dmonte, Roland Oruche, Marcos Zampieri, Prasad Calyam, and Isabelle Augenstein. 2025. Claim Verification in the Age of Large Language Models: A Survey. doi:10.48550/arXiv.2408.14317 arXiv:2408.14317.
- [11] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. RAGAs: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Nikolaos Aletras and Orpheus De Clercq (Eds.). Association for Computational Linguistics, St. Julians, Malta, 150–158. doi:10.18653/v1/2024.eacl-demo.16
- [12] Feng Fu, Hong Ding, Yong Qin, Jian Yu, and Dechao Xu. 2025. Leveraging multi-agent framework for root cause analysis. *Complex & Intelligent Systems* 12, 1 (Nov. 2025), 4. doi:10.1007/s40747-025-02096-0
- [13] Ruowei Fu, Yang Zhang, Zeyu Che, Xin Wu, Zhenyu Zhong, Zhiqiang Ren, Shenglin Zhang, Feng Wang, Yongqian Sun, Xiaozhou Liu, Kexin Liu, and Yu Zhang. 2025. LLM-Powered Multi-Agent Collaboration for Intelligent Industrial On-Call Automation. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*. IEEE, 2222–2234.
- [14] Luyu Gao, Zhu Yun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. 2023. RARR: Researching and Revising What Language Models Say, Using Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 16477–16508. doi:10.18653/v1/2023.acl-long.910
- [15] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling Deep Contrastive Learning Batch Size under Memory Limited Setup. In *Proceedings of the 6th Workshop on Representation Learning for NLP, RepL4NLP@ACL-IJCNLP 2021*. Association for Computational Linguistics, 316–321.
- [16] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. 2026. A survey on LLM-as-a-Judge. *The Innovation* (Jan. 2026), 101253. doi:10.1016/j.xinn.2025.101253
- [17] Xiao Han, Shuhan Yuan, and Mohamed Trabelsi. 2023. LogGPT: Log Anomaly Detection via GPT. In *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*. IEEE, 1117–1122.
- [18] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.
- [19] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xianwu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=VtmBAGCN7o>

- [20] Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2024. FaultProfiT: Hierarchical Fault Profiling of Incident Tickets in Large-scale Cloud Systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. ACM, Lisbon Portugal, 392–404. doi:10.1145/3639477.3639754
- [21] Shashidhar Reddy Javaji, Yupeng Cao, Haohang Li, Yangyang Yu, Nikhil Muralidhar, and Zining Zhu. 2025. Can AI Validate Science? Benchmarking LLMs on Claim → Evidence Reasoning in AI Papers. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dharendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 2355–2379. doi:10.18653/v1/2025.ijcnlp-long.127
- [22] Tharushi Jayasekara and Immanuel Trummer. 2025. Demonstrating CEDAR: A System for Cost-Efficient Data-Driven Claim Verification. In *Companion of the 2025 International Conference on Management of Data*. ACM, Berlin Germany, 135–138. doi:10.1145/3722212.3725098
- [23] Saurabh Jha, Rohan R. Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O. Turkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Jae-Wook Ahn, Debanjana Kar, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav R. Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas C. M. Fuller, and Ruchir Puri. 2025. ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 27134–27197. <https://proceedings.mlr.press/v267/jha25a.html>
- [24] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. An Approach for Anomaly Diagnosis Based on Hybrid Graph Model with Logs for Distributed Services. In *2017 IEEE International Conference on Web Services, ICWS 2017*. IEEE, 25–32.
- [25] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Xpert: Empowering Incident Management with Query Recommendations via Large Language Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–13. doi:10.1145/3597503.3639081
- [26] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Xpert: Empowering Incident Management with Query Recommendations via Large Language Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–13. doi:10.1145/3597503.3639081
- [27] Carlos E. Jimenez, John Yang, Alexander Wetteg, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=VTF8yNQm66>
- [28] Pedro Las-Casas, Alok Kumbhare, Rodrigo Fonseca, and Sharad Agarwal. 2024. LLexus: an AI agent system for incident management. (Aug. 2024). <https://www.microsoft.com/en-us/research/publication/llexus-an-ai-agent-system-for-incident-management/>
- [29] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Thirty-seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=3lyL2XWDkG>
- [30] Fengrui Liu, Xiao He, Tieying Zhang, Jianjun Chen, Yi Li, Lihua Yi, Haipeng Zhang, Gang Wu, and Rui Shi. 2025. TickIt: Leveraging Large Language Models for Automated Ticket Escalation. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. ACM, Clarion Hotel Trondheim Trondheim Norway, 343–354. doi:10.1145/3696630.3728558
- [31] Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 9004–9017. doi:10.18653/v1/2023.emnlp-main.557
- [32] Grégoire Mialon, Clémentine Fourier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for General AI Assistants. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=fbxvahvs3>
- [33] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. MemGPT: Towards LLMs as Operating Systems. doi:10.48550/arXiv.2310.08560 arXiv:2310.08560.
- [34] Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. TALM: Tool Augmented Language Models. doi:10.48550/arXiv.2205.12255 arXiv:2205.12255.
- [35] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, San Francisco CA USA, 1–22. doi:10.1145/3586183.3606763
- [36] 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. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis. In *Companion Proceedings of the ACM on Web Conference 2025*. ACM, Sydney NSW Australia, 422–431. doi:10.1145/3701716.3715225
- [37] Binpeng Shi, Yu Luo, Jingya Wang, Yongxin Zhao, Shenglin Zhang, Bowen Hao, Chenyu Zhao, Yongqian Sun, Zhi Zhang, Ronghua Sun, Haihua Li, Wei Song, Xiaolong Chen, Jingbo Miao, and Dan Pei. 2025. FlowXpert: Expertizing Troubleshooting Workflow Orchestration with Knowledge Base and Multi-Agent Coevolution. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*. ACM, Toronto ON Canada, 4839–4850. doi:10.1145/3711896.3737221
- [38] Yongqian Sun, Bowen Hao, Xiaotian Wang, Chenyu Zhao, Yongxin Zhao, Binpeng Shi, Shenglin Zhang, Qiao Ge, Wenhua Li, Hua Wei, and Dan Pei. 2025. LLM-Augmented Ticket Aggregation for Low-cost Mobile OS Defect Resolution. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. ACM, Clarion Hotel Trondheim Trondheim Norway, 215–226. doi:10.1145/3696630.3728547

- [39] Yongqian Sun, Bowen Hao, Xiaotian Wang, Chenyu Zhao, Yongxin Zhao, Binpeng Shi, Shenglin Zhang, Qiao Ge, Wenhui Li, Hua Wei, and Dan Pei. 2025. LLM-Augmented Ticket Aggregation for Low-cost Mobile OS Defect Resolution. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. ACM, Clarion Hotel Trondheim Trondheim Norway, 215–226. doi:10.1145/3696630.3728547
- [40] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. MAGIS: LLM-based multi-agent framework for GitHub issue ReSolution. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA. event-place: Vancouver, BC, Canada.
- [41] Mohamed Trabelsi and Ahmet Akyamac. 2025. Learning from Imperfect Data for Anomaly Detection. U.S. Patent Application US20250384261A1. Published Dec. 18, 2025.
- [42] Mohamed Trabelsi and Huseyin Uzunalioglu. 2023. Absformer: Transformer-Based Model for Unsupervised Multi-Document Abstractive Summarization. In *Document Analysis and Recognition - ICDAR 2023 Workshops, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14194)*. Springer, 151–166.
- [43] Mohamed Trabelsi and Huseyin Uzunalioglu. 2026. TeleDoCTR: Domain-Specific and Contextual Troubleshooting for Telecommunications. doi:10.48550/arXiv.2601.00691 arXiv:2601.00691.
- [44] Haoran Wang, Aman Rangapur, Xiong Xiao Xu, Yueqing Liang, Haroon Gharwi, Carl Yang, and Kai Shu. 2025. Piecing It All Together: Verifying Multi-Hop Multimodal Claims. In *Proceedings of the 31st International Conference on Computational Linguistics*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, Abu Dhabi, UAE, 7453–7469. <https://aclanthology.org/2025.coling-main.498/>
- [45] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 2609–2634. doi:10.18653/v1/2023.acl-long.147
- [46] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, Dongmei Zhang, Changhua Pei, and Gaogang Xie. 2024. Large Language Models Can Provide Accurate and Interpretable Incident Triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Tsukuba, Japan, 523–534. doi:10.1109/ISSRE62328.2024.00056
- [47] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. ACM, Boise ID USA, 4966–4974. doi:10.1145/3627673.3680016
- [48] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryan W. White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *First Conference on Language Modeling*. <https://openreview.net/forum?id=BAakY1hNKS>
- [49] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-Mem: Agentic Memory for LLM Agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=FiM0M8gcct>
- [50] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=mXpq6ut8J3>
- [51] Zhaoyang Yu, Aoyang Fang, Minghua Ma, Jaskaran Singh Walia, Chaoyun Zhang, Shu Chi, Ze Li, Murali Chintalapati, Xuchao Zhang, Rujia Wang, Chetan Bansal, Saravan Rajmohan, Qingwei Lin, Shenglin Zhang, Dan Pei, and Pinjia He. 2025. Triangle: Empowering Incident Triage with Multi-Agent. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16–20, 2025*. IEEE, 674–686. doi:10.1109/ASE63991.2025.00062
- [52] Boya Zhang, Alban Bornet, Anthony Yazdani, Philipp Khlebnikov, Marija Milutinovic, Hossein Rouhizadeh, Poorya Amini, and Douglas Teodoro. 2025. A dataset for evaluating clinical research claims in large language models. *Scientific Data* 12, 1 (Jan. 2025), 86. doi:10.1038/s41597-025-04417-x
- [53] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Yan Chaoran, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, and Bo Zhang. 2024. mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 4017–4033. doi:10.18653/v1/2024.findings-emnlp.232
- [54] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. <https://openreview.net/forum?id=uclHPGDlao>
- [55] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and others. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854* (2023). <https://webarena.dev>

A ASTRA Prompts

This appendix reproduces the system-level instruction prompts from ASTRA's pipeline. The claim extraction, evidence mapping, and orchestrator planning prompts also appear inline in Section 4 alongside the corresponding pipeline descriptions. All LLM calls use gpt-oss:120b at temperature 0 unless otherwise noted. Prompts marked with variable sections (e.g., ticket context, log content, agent outputs) show the static instruction portions; dynamic content is assembled at runtime by the orchestration code.

A.1 TicketSimilarityAgent Analysis Prompts

The TicketSimilarityAgent uses a two-phase LLM reranker to select the most diagnostically useful historical tickets from candidates retrieved by dense vector search. Phase 1 processes batches of 20 candidates each; Phase 2 makes the final selection from the Phase 1 survivors.

Phase 1 Batch Reranker System Message:

You are a Nokia 5G/LTE RAN fault analyst reviewing historical tickets.

A ticket is USEFUL for the target if reading its Identification and Resolution would meaningfully help an engineer diagnose or fix the target fault. Concretely:

- The root cause operates in the same SW component or subsystem
- The failure mechanism (what broke and why) is directly comparable
- The resolution gives actionable information: a specific parameter, flag, function, or configuration change relevant to the target

A ticket is NOT useful if:

- Its root cause is in a different subsystem (different SW layer, different feature)
- The root cause is blank, "not defined", "unknown", or too vague
- The only connection is a shared surface keyword with nothing transferable about how or why it failed

Note: embedding similarity is shown for context only -- it does not indicate usefulness.

Respond ONLY with a valid JSON object:

```
{
  "selected": [{"pr_id": "<id>", "usefulness": "HIGH|MEDIUM|LOW",
    "reason": "<one sentence>"}],
  "excluded_reason": "<one sentence>"}

```

Select at most 5. Fewer or empty list is valid.

Phase 2 Final Selection System Message:

You are a senior Nokia 5G/LTE RAN fault analyst making the final selection of historical tickets to anchor a root cause analysis.

Apply the same usefulness test as Phase 1. Additional guidance:

- Prefer tickets with specific, confirmed root causes over vague ones
- If several tickets describe the exact same fault mechanism, pick the one with the most complete and actionable resolution; exclude the rest as duplicates
- Diverse coverage is better than five tickets all pointing to the same hypothesis
- A wrong-domain ticket in the final 5 actively harms downstream analysis -- when in doubt, exclude it

You may agree or override Phase 1 usefulness ratings.

Respond ONLY with a valid JSON object:

```
{
  "final_selection": [{"pr_id": "<id>", "rank": "<1-5>",
    "usefulness": "HIGH|MEDIUM|LOW", "reason": "<1-2 sentences>"}],

```

```
"excluded": {"<pr_id>": "<one-sentence reason>"},
"summary": "<2 sentences: what fault domain do the selected tickets collectively point to>"
Rank 1 = most useful. At most 5.
```

Both phases receive a user message containing the target ticket metadata (product, feature, build, title, description) and the candidate block with ticket details and similarity scores.

A.2 LogAgent Analysis Prompts

The LogAgent issues a single constrained LLM call. The prompt template assembles: ticket context, enhanced description, acronym expansions, component context, an optional similar-ticket reference block (shown below), and the filtered log content. The system message and appended analysis task are:

```
LogAgent Prompt
-----
System message:
You are a precise log evidence extractor for wireless network equipment.
You ONLY report what you can directly see in the provided log lines.
You NEVER invent details not present in the logs.
When in doubt, say 'not determinable from logs'.

Analysis task (appended after log lines):
Analyze ONLY the log lines provided above. Extract factual observations.

STRICT RULES:
1. EVERY finding must directly quote at least one log line from above.
2. DO NOT invent file paths, function names, or error codes not in the logs.
3. DO NOT speculate about code-level root causes unless the log says so.
4. If root cause is unclear, say "Root cause not determinable from logs alone".
5. Focus on WHAT happened (observable facts) not WHY (speculation).

Return JSON:
{
  "findings": [ { "type", "component", "log_quote", "timestamp",
                  "frequency", "severity", "description",
                  "fault_relevance", "relevance_reasoning" } ],
  "temporal_sequence": "...",
  "root_cause_assessment": {
    "from_logs": "...", "confidence": "high|medium|low|none",
    "needs_further_investigation": "..."
  },
  "summary": "2-3 sentence summary based ONLY on log evidence"
}
```

When similar tickets are available from the pre-fetch, the following reference block is prepended to the log lines section:

```
Similar Tickets Reference Block
-----
SIMILAR RESOLVED HISTORICAL TICKETS [LOG INTERPRETATION GUIDE]
Use these confirmed resolutions to:
1. Classify log findings: same component/error pattern = likely CAUSAL
2. Identify ambiguous log token owners
3. Confirm root cause plausibility -- if logs match the reference pattern, say so
Do NOT copy the resolution verbatim. Only include claims corroborated by log lines above.
```

```
[HIGH/GOOD/POSSIBLE MATCH] Reference N: <ticket title>
Root cause: <root cause text up to 200 chars>
Resolution: <resolution summary up to 300 chars>
Investigation trail: <TA trail for top-2 ranked tickets, up to 400 chars>
```

A.3 DomainKnowledgeAgent Analysis Prompts

The DomainKnowledgeAgent queries technical documentation via MCP (Model Context Protocol). It uses a system message that constrains the response to fault-relevant content, and an enriched task query constructed from the ticket context and other agents' outputs.

System Message:

```
You are a technical documentation assistant for telecom network equipment.
Focus your answers on fault-relevant content: root cause explanations, error code meanings, configuration parameters, and
resolution procedures.
Cite the specific document or section when available.
If no retrieved document directly addresses the query, state that clearly rather than generalizing.
Structure your response to distinguish between identification-relevant findings (symptoms, error meanings, diagnostic
procedures) and resolution-relevant findings (fixes, configuration changes, workarounds).
```

Enriched Task Query (constructed at runtime):

```
Find documentation for this Nokia fault -- root cause, known issues, and resolution: {ticket_title} [Alarm IDs: {
extracted_FIDs}].
Focus on: (1) what triggers this alarm, (2) configuration parameters and correct values, (3) step-by-step resolution
procedure.

CONTEXT FROM OTHER AGENTS (use to narrow your documentation search):
Log analysis findings:
{logagent_output[:2000]}

Similar historical tickets (likely fault domain):
{tsa_output[:1500]}
```

The agent output context is truncated (2,000 characters from LogAgent, 1,500 from TicketSimilarityAgent) to stay within the documentation retrieval system's query limits.

A.4 Report Generation Prompt

This prompt is used for both the initial report and each revision pass. During revision, the judge's feedback and new agent findings from the follow-up round are appended before the claim-evidence block.

```
Report Generation Prompt
-----
You are a telecom fault analysis expert. Generate a fault analysis report using the provided claim-evidence mapping.

The report has exactly two sections:

identification -- How was the fault detected and diagnosed?
Required: initial alarm/symptom -> diagnostic steps -> causal chain -> root cause.
Include: FID/alarm IDs, exact error messages, configuration parameters, log citations, and the name of the affected
module or protocol layer.
```

```

resolution -- How was the fault resolved?
  Required: exact fix applied -> specific parameter values changed -> verification.
  If a workaround was applied instead of a permanent fix, state that explicitly.

Source attribution:
  For every technical claim, state which agent provided it.
  [VERIFIED] claims are directly grounded -- cite as facts.
  [UNVERIFIED] claims: prefix with "Analysis suggests (unverified)..." -- do NOT state specific IDs or values from
    unverified claims as confirmed facts.

Quality constraints:
  - 300-600 words per section; completeness takes priority over length targets.
  - Do NOT repeat information -- if LogAgent and TSA both describe the same root cause, merge them, citing both.
  - NEVER invent specific values (IDs, file paths, parameters, timestamps) absent from the evidence.

TicketSimilarityAgent conflict rule:
  If TSA matches describe a different fault domain than LogAgent findings, do NOT follow the TSA match. State the conflict
    and give log evidence priority.

Return JSON: { "identification": "...", "resolution": "..."}

```

A.5 Judge Evaluation Prompt

The judge evaluates each generated report on a 1–5 scale across five criteria. A score ≤ 3 on any criterion sets `needs_revision: true` and triggers a follow-up round.

```

Judge Evaluation Prompt
-----
You are a Quality Assurance Judge evaluating a Fault Analysis Report (FA) for technical accuracy and factual grounding.
  You are given: (1) agent outputs, (2) the claim-evidence mapping, (3) the FA report, (4) the original ticket context.

Goal: verify the FA is grounded in the evidence and identify improvements.
You do NOT have access to ground truth -- evaluate only against what agents found.

METRIC 1: Root Cause Identification
  5 -- Precise root cause (specific component + failure mode), corroborated by >=2 sources.
  4 -- Specific root cause, supported by one strong agent source.
  3 -- General fault area described; exact mechanism not pinpointed from evidence.
  2 -- Vague throughout, or names component absent from all agent outputs.
  1 -- No root cause, or directly contradicts agent reports.
  Red flag (score <=2): FA asserts root cause mechanism untraceable to any agent output.

METRIC 2: Resolution Specificity
  5 -- Every step cites an agent source with exact identifiers.
  4 -- Most steps specific; at most one generic step.
  3 -- Correct fix type described but vaguely -- missing specific parameters/values.
  2 -- Generic advice with no ticket-specific details.
  1 -- Recommends actions absent from or contradicted by all agent outputs.

METRIC 3: Evidence Grounding
  5 -- >=90% of specific identifiers appear verbatim in agent outputs.
  4 -- 70-90% grounded; minor unverified claims clearly marked.
  3 -- 50-70% grounded -- several non-trivial claims lack direct evidence.
  2 -- <50% grounded.
  1 -- FA invents specific identifiers not present in any agent output.

METRIC 4: Coverage & Completeness
  5 -- All four dimensions (symptom, root cause, fix, verification) with evidence.

```

```

4 -- All four present; one has limited depth.
3 -- Three dimensions present; one missing or placeholder.
2 -- Only two dimensions addressed.
1 -- Only symptom description; root cause and resolution absent.
METRIC 5: Focus & Relevance
5 -- Every clause directly addresses the ticket fault.
4 -- Mostly on-topic; one or two minor tangents.
3 -- Partially distracted by off-topic evidence.
2 -- Significant portion (>=1 paragraph) built around unrelated evidence.
1 -- Narrative anchored to evidence unrelated to reported symptoms.

For any metric scoring <=3, set improvement_needed: true and specify: suggested_focus: what is missing or could be made
    more specific recommended_agents: which agent is most likely to supply the missing evidence
Set needs_revision: true if ANY metric scores <=3.
Set revision_priority to the single most impactful metric to strengthen first.

Return a single valid JSON object with all scores and improvement recommendations.

```

A.6 TA-Aware Evaluation Prompt

This prompt is used by the external evaluation judge (Section 5) to score each final ASTRA-generated report against both the ground-truth FA and the engineer TA investigation chain. It produces the five reported metric scores in Table 1 which are Accuracy, Completeness, Relevance, Clarity, and TechnicalDepth. The evaluation judge is distinct from the pipeline JudgeAgent (Appendix A.5): the pipeline judge scores against *agent outputs* during generation to trigger refinement, whereas this evaluation judge scores against *engineer references* to assess final quality.

```

TA-Aware Evaluation Judge Prompt
-----
You are evaluating a predicted fault analysis (FA) report for a telecom software defect ticket. You have TWO reference
sources:
1. The official Ground Truth FA (concise engineer summary)
2. The Technical Analysis (TA) investigation chain (multi-round engineer debugging notes)

IMPORTANT CONTEXT: The predicted FA was generated by an automated AI system that works from log files, similar historical
tickets, and technical documentation. It did NOT have access to the TA investigation chain or the live debugging
sessions that engineers performed. Evaluate what the system achieves from its available evidence sources -- do not
penalize it for lacking information that is only obtainable through live debugging or internal team hand-offs.

--- TICKET INFORMATION ---
<ticket metadata and description>

--- GROUND TRUTH FAULT ANALYSIS (official FA template) ---
Concise engineer-written summary. Often sparse but captures the final root cause category and fix direction.
<ground truth text>

--- TECHNICAL ANALYSIS CHAIN (engineer investigation notes) ---
Multiple analysis rounds from engineers during live debugging. Contains raw log snippets, specific parameter names,
component names, and final conclusions. Use this as supplementary context to better understand the fault -- but
remember the AI system never saw this data.
<TA chain text>

--- PREDICTED FAULT ANALYSIS ---
Produced by a multi-agent AI system using log analysis, similar ticket retrieval, and documentation queries. Has two
sections: IDENTIFICATION and RESOLUTION.

```

```

<generated FA report>

--- EVALUATION ---
Evaluate the predicted FA on these 5 metrics. Score each from 0-5.

1. **Accuracy** -- Does the predicted FA identify the same fault domain as the GT? Evaluate in two layers: (a) symptom/
    fault-area alignment, then (b) root cause mechanism alignment. Use the TA to understand the full picture, but
    evaluate primarily based on whether the prediction is directionally consistent.
    - Score 5: Same symptom AND same specific root cause mechanism as GT/TA
    - Score 4: Same symptom AND same root cause direction, minor differences
    - Score 3: Same symptom AND same fault area/component family, but different specific root cause mechanism or code path
    - Score 2: Related product domain but different fault area, OR correct symptom but entirely fabricated cause
    - Score 1: Different fault domain entirely (wrong subsystem, wrong symptom category)
    - Score 0: Completely unrelated fault

2. **Completeness** -- Does the prediction address all four expected fault analysis dimensions: (a) symptom/alarm
    detection, (b) root cause analysis, (c) resolution/fix steps, (d) verification or preventive measures?
    Do NOT penalize for missing TA-specific details the AI system could not access; evaluate whether the prediction covers
    the essential diagnostic dimensions from its own evidence.

3. **Relevance** -- Does the prediction stay focused on this specific ticket? No off-topic information.

4. **Clarity** -- Is the predicted FA well-structured and easy to follow for a technical engineer?

5. **TechnicalDepth** -- Does the FA include specific technical details grounded in evidence (parameter names, error codes,
    log excerpts, alarm IDs, component identifiers, timestamps)? Details that also appear in the TA score highest, but
    specific details derived from log analysis or similar tickets are also valuable.
    - Score 5: Cites specific identifiers and multiple also appear in TA
    - Score 4: Includes concrete technical details grounded in log/ticket evidence; some align with TA findings
    - Score 3: Mentions the right subsystem and symptom with some specific details from its sources
    - Score 2: Mostly generic but includes one or two specific details
    - Score 1: Generic description only -- could apply to any similar ticket
    - Score 0: Completely generic or fabricated technical details

ACCURACY DECISION RULE: If the prediction correctly identifies the symptom AND the affected subsystem/component family,
    score Accuracy >= 3 regardless of whether the specific root cause mechanism matches. Score 2 requires the fault AREA
    (not just the mechanism) to be wrong. Score 1 requires the fault DOMAIN to be wrong.

Key principles:
- The prediction was generated WITHOUT access to the TA chain. Do not require TA-level specificity for scores 3-4.
- A detailed, evidence-grounded prediction that is directionally consistent with GT should score 3-4 on Accuracy even if
    it reaches a slightly different specific mechanism than TA.
- Reserve scores 0-2 for predictions where the fault domain or component is fundamentally wrong.
- Generic boilerplate answers score low on TechnicalDepth even if directionally correct.

Return a JSON object:
{
  "Accuracy": {"analysis_and_justification": "<explanation>", "score": <0-5>},
  "Completeness": {"analysis_and_justification": "<explanation>", "score": <0-5>},
  "Relevance": {"analysis_and_justification": "<explanation>", "score": <0-5>},
  "Clarity": {"analysis_and_justification": "<explanation>", "score": <0-5>},
  "TechnicalDepth": {"analysis_and_justification": "<explanation>", "score": <0-5>}
}

```