Context-Aware Multi-Agent Root Cause Analysis for Distributed Transaction Failures

Veera Ravindra Divi
Independent Researcher
Austin, TX, USA

ABSTRACT

A single distributed transaction can touch a dozen services, several databases, an authorization layer, a message bus, and one or two third-party dependencies before it either commits or fails. When it fails, the engineer on call rarely sees the cause directly. They see a symptom—a 500 returned to a caller, a stalled queue, a customer ticket—and then spend the bulk of their time reconstructing the transaction path, deciding which team owns the failing hop, correlating logs across service boundaries, and separating an upstream cause from an internal bug from a downstream outage. This paper presents CARMA, a Context-Aware Root-cause Multi-Agent Analysis framework that decomposes this work across specialized agents: a ticket-understanding agent, a log-retrieval agent, a dependency-classification agent, a payload-analysis agent, a policy-checking agent, and a remediation-recommendation agent, coordinated by an orchestrator. The framework's central idea is evidence-weighted aggregation: each agent emits scored, citable evidence rather than a verdict, and the orchestrator resolves a root-cause class from the weighted set, which keeps the final answer grounded in retrieved signals rather than free generation. A synthetic benchmark is defined that injects seven representative failure classes—malformed request, authorization mismatch, cache inconsistency, timeout propagation, dependency unavailability, schema drift, and business-rule rejection—into generated transaction graphs, together with an evaluation protocol spanning classification accuracy, mean-time-to-diagnosis, false escalation rate, remediation precision, human-review workload, latency, and cost. All reported numbers are illustrative figures produced by the simulation harness and are intended to characterize the framework's behavior, not to report production performance.

General Terms

Reliability, Diagnosis, Agents

Keywords

Root cause analysis, multi-agent systems, distributed transactions, observability, incident response

1. INTRODUCTION

At 02:14 on a weekday, a payment-authorization transaction begins to fail intermittently. The customer-facing symptom is a generic 500 with a retry banner. The on-call engineer opens the incident ticket and finds one sentence: "checkout failing for some users." They pull the trace by its trace id and see a span tree that fans out across an API gateway, an order service, a pricing service, a fraud-scoring dependency, and a ledger write. One span is marked ERROR, but the error message is a wrapped timeout that says nothing about which downstream hop stalled first. The engineer opens five log dashboards, each owned by a different team, and begins stitching: matching the trace id against log lines, lining up timestamps that are skewed by a few hundred milliseconds, and guessing whether the fraud dependency timed out, whether the order service exhausted its retry budget waiting on it, or whether a recent schema change in the pricing payload quietly broke deserialization. Forty minutes later the answer turns out to be mundane: a downstream dependency tightened a response field, the order service's deserializer rejected the new shape, and the rejection surfaced as a timeout because the client-side circuit breaker tripped on the resulting latency. The diagnosis was slow not because the cause was deep, but because the evidence was scattered across boundaries that no single person owns.

This pattern repeats across organizations that run service meshes. The hard part of distributed root cause analysis is rarely a single missing fact; it is the integration of many partial facts that live in different systems, are owned by different teams, and arrive in different formats. Traces tell the shape of the call graph and where time was spent, but not why a payload was rejected. Logs tell what a service decided, but not how that decision relates to the request that started upstream. Metrics tell that error rates rose, but not which of three plausible causes is responsible. An engineer is effective precisely because they fuse these signals; that fusion is also the slowest and least scalable part of incident response, and it is the part that gets worse as the dependency graph grows.

Recent work in agentic artificial intelligence suggests a structure for this problem. Language-model agents that reason and act in a loop can retrieve evidence on demand [1], decompose a problem into intermediate steps [2], ground their output in retrieved context rather than parametric memory [3], and coordinate as a team of specialists [4]. The temptation is to point a single large model at an incident and ask it to "find the root cause." In practice a single agent conflates retrieval, reasoning, and recommendation, which makes its output hard to audit and prone to confidently asserting a cause that no retrieved signal supports. The diagnostic task has natural seams—understand the ticket, fetch the logs, classify the failing dependency edge, inspect the payload, check the policy, recommend a fix—and those seams suggest a division of labor rather than a monolith.

This paper proposes CARMA (Context-Aware Root-cause Multi-Agent Analysis), a framework that assigns each seam to a specialized agent and coordinates them through an orchestrator. The framework's distinguishing idea is that agents do not vote on a verdict; they emit scored, citable evidence, and the orchestrator computes an evidence-weighted root-cause score across the candidate classes. This keeps the framework grounded: a root-cause class can only win if at least one agent produced a concrete signal—a log line, a span attribute, a payload diff, a policy clause—that supports it, which bounds (though it does not eliminate) the framework's tendency to hallucinate causes. The framework is instantiated against a synthetic benchmark of seven representative failure classes and an evaluation protocol that measures both diagnostic quality and operational cost.

This paper makes three contributions. First, it presents a multi-agent decomposition of distributed transaction root cause analysis in which each agent is specified by responsibility, inputs, outputs, failure modes, and controls, and it gives an

evidence-weighted aggregation rule that turns scored agent evidence into a single grounded root-cause decision. Second, it defines a failure-mode taxonomy of seven classes and an original signal-mapping that records, for each class, which observability signal each agent is expected to contribute—making the framework's reasoning legible and its evaluation falsifiable. Third, it describes a synthetic benchmark generator and an evaluation protocol spanning classification accuracy, MTTD, false escalation rate, remediation precision, human-review workload, latency, and cost, and it reports illustrative results that characterize the trade-offs the framework exposes. The remainder of the paper is organized as follows. Section 2 fixes terminology. Section 3 surveys related work grouped by limitation. Section 4 states the problem formally. Section 5 describes CARMA. Section 6 details the evaluation design. Section 7 reports illustrative results. Sections 8 through 11 discuss trade-offs, threats to validity, practical implications, and conclusions.

2. BACKGROUND AND MOTIVATION

A distributed transaction, in the sense used here, is a unit of business work—authorize a payment, place an order, provision a resource—whose completion requires coordinated calls across two or more independently deployed services. The strict ACID guarantees of a database transaction are not required; most production systems relax those guarantees in favor of sagas, idempotent retries, and compensating actions. What matters for diagnosis is that the work spans ownership and trust boundaries, so no single service has a complete view of why the transaction failed.

A partial failure is the characteristic failure mode of such systems: some hops succeed and some fail, and the surviving symptom at the edge is often disconnected from the originating fault. A downstream dependency that returns slowly can manifest as an upstream timeout; a malformed field in a request can manifest as a generic deserialization error three services away; a stale cache entry can produce a correct-looking response that fails a later consistency check. The engineer's job is to trace the symptom back to the originating hop and to attribute ownership correctly, because the team that sees the alarm is frequently not the team that must fix the cause.

Three families of observability signals support that work. Logs are timestamped, service-local records of decisions and errors; they are rich but unstructured and siloed. Metrics are aggregated time series—error rates, latency percentiles, saturation—that reveal that something changed but rarely why. Traces record the causal path of a single request as a tree of spans, each with a duration and attributes, tied together by a trace id propagated across services [5]. The W3C Trace Context specification standardizes how that identifier is carried over service boundaries [6], and the OpenTelemetry specification defines a common data model for emitting all three signal types [7]. The reliability of that telemetry is itself a variable that gates downstream diagnosis, which has motivated frameworks that score telemetry quality as an AI-readiness signal for automated operations [19]. The promise of this telemetry is exactly the context CARMA needs; the reality is that the signals remain unintegrated until a human—or an agent—fuses them.

The relevant operational target is mean-time-to-resolution and, within it, the time spent on diagnosis as opposed to repair. Site reliability engineering practice treats this diagnostic latency as a primary cost driver in incident response [8], and software-quality standards frame reliability and recoverability as first-class product attributes [9]. By agentic AI is meant systems in which a language model is given tools and a loop: it observes, decides on an action such as a retrieval, executes it, observes the result, and iterates [1]. The reason this paradigm fits diagnosis is that diagnosis is itself iterative evidence-gathering; the reason it must be handled carefully is that a model asked to explain a failure will produce a plausible explanation whether or not the evidence supports it, and in an operational setting a confident wrong attribution is worse than an honest "insufficient evidence."

3. RELATED WORK

Prior work is grouped by the limitation that motivates CARMA rather than by technique, because the framework's contribution is integrative.

Rule-based alerting and triage. The dominant production approach encodes operator knowledge as static rules: if error rate exceeds a threshold on service X, page team X; if a specific exception string appears, route to a runbook. Stability patterns such as the circuit breaker and bulkhead, popularized by Nygard [10], are likewise rule-shaped responses to known failure modes. These systems are fast, cheap, and explainable, and they remain the right tool for recurring, well-understood faults. Their limitation is brittleness: they do not fuse context across signals, they attribute to the alarming service rather than the originating one, and they degrade sharply on novel or cross-boundary failures, which is precisely the regime that consumes engineer time.

Trace-based and causality-graph RCA. A second line builds an explicit causal or dependency structure from telemetry and localizes faults within it. CauseInfer constructs a hierarchical causality graph to pinpoint performance problems [11]; failure sketching reconstructs the causal chain of an in-production failure for diagnosis [12]. This work fuses context well and attributes causally, drawing on the formal theory of causal inference [13]. Its limitation is that it typically operates on a single signal type and produces a localization rather than a remediation; it tells where but not what to do, and it rarely incorporates the natural-language context—tickets, policies, payload schemas—that disambiguates similar-looking faults.

Machine-learning anomaly detection and AIOps. A third line learns normal behavior from trace logs or metrics and flags deviations, sometimes localizing the responsible service [14]. Surveys of AIOps catalog these methods and their move toward language-model-based variants [15]. These methods scale to high signal volume and detect novel anomalies, but they are weak on explainability—a flagged anomaly is not an attributed cause—and they generally do not recommend remediations or reason about ownership.

Single-LLM incident assistants. The newest line applies a single large language model to incident summaries, drawing on reasoning [2], retrieval grounding [3], and the transformer architecture underneath [16]. A single model can fuse heterogeneous context and produce both an attribution and a remediation in natural language. Its limitation is auditability and hallucination: when retrieval, reasoning, and recommendation collapse into one opaque pass, it is hard to tell which part of the output is evidence-backed, and the model can assert a cause no signal supports. Multi-agent conversation frameworks [4] provide a structure to separate these concerns, and a multi-agent debate framework with an evidence-consolidating arbiter and a mandatory human approval gate has been applied to architecture governance in the same spirit [18]; risk-management guidance for AI systems recommends exactly this kind of traceability and human oversight [17]. CARMA sits in this last cell: it keeps the context fusion and remediation of a single LLM while restoring auditability through specialized agents and evidence-weighted aggregation. Table 1

summarizes the comparison.

Table 1. Related approaches grouped by limitation. "Context fusion" is the ability to combine logs, traces, and natural-language context; "human workload" is qualitative residual effort left to the engineer.

Approach	Context fusion	Causal attribution	Remediation	Explainability	Human workload
Rule-based triage	None	Alarming service	Runbook link	High	High on novel faults
Trace/causality RCA	Single signal	Strong (localization)	None	Medium	Medium
ML / AIOps anomaly	Single signal	Weak (flag only)	None	Low	Medium
Single-LLM assistant	Strong	Opaque	Yes	Low	Low but risky
CARMA (this work)	Multi-signal	Evidence-weighted	Yes, scoped	High (citable)	Low, bounded

4. PROBLEM DEFINITION

One diagnosis episode is formalized. Let a failed distributed transaction be described by a tuple (T, L, Q, G). The trace T = (V, E_T) is a directed tree of spans V rooted at the edge-facing request; each span v ∈ V carries a service identifier, start and end timestamps, a status in {OK, ERROR}, and an attribute map. The log bundle L = {ℓ₁, …, ℓ_m} is the set of log records correlated to the transaction by trace id, each with a source service, a timestamp, a severity, and free text. The ticket Q is the natural-language incident description, typically terse and symptom-level. The dependency graph G = (S, E_G) is the static service topology, where nodes S are services with declared owners and edges E_G are call relationships annotated with timeout and retry-budget configuration.

The diagnostic function must produce a triple (c, o, r), where c ∈ C is a root-cause class drawn from a fixed taxonomy (Section 5.3), o ∈ S is the owning service of the originating fault, and r is a remediation recommendation. A confidence κ ∈ [0,1] and an evidence set X of citable references into T, L, Q, or G that justify (c, o, r) are additionally required.

A diagnosis is correct when the predicted class matches the injected ground-truth class, ĉ = c*, and the predicted owner matches the originating service, ô = o*. Class and owner are treated jointly because attributing the right failure type to the wrong team is operationally almost as costly as missing the type. Three failure conditions of the diagnostic function itself are distinguished. A misclassification returns the wrong c or o with high confidence. A false escalation returns κ below the auto-resolution threshold τ and routes the incident to a human when the framework in fact held sufficient evidence to resolve it; this wastes the workload reduction the framework exists to provide. An ungrounded diagnosis returns a (c, o, r) whose evidence set X is empty or does not actually support the claim. The framework's design goal is to minimize misclassification and ungrounded diagnoses while keeping false escalations low enough that the residual human workload is meaningfully smaller than the rule-based baseline.

5. THE CARMA FRAMEWORK

5.1 Overview

CARMA decomposes the diagnostic function into six specialized agents and an orchestrator, arranged over the transaction's dependency graph (Figure 1). Each agent is a language model equipped with a narrow tool set and a single responsibility. Crucially, agents do not emit verdicts; each emits a set of scored evidence items, where an item is a tuple (root-cause class, supporting reference, agent confidence). The orchestrator collects these items and computes an evidence-weighted score per candidate class. This separation is the framework's main design commitment: it keeps every contribution to the final decision traceable to a specific signal and a specific agent, which is what makes the output auditable in the sense recommended for AI systems used in consequential settings [17].

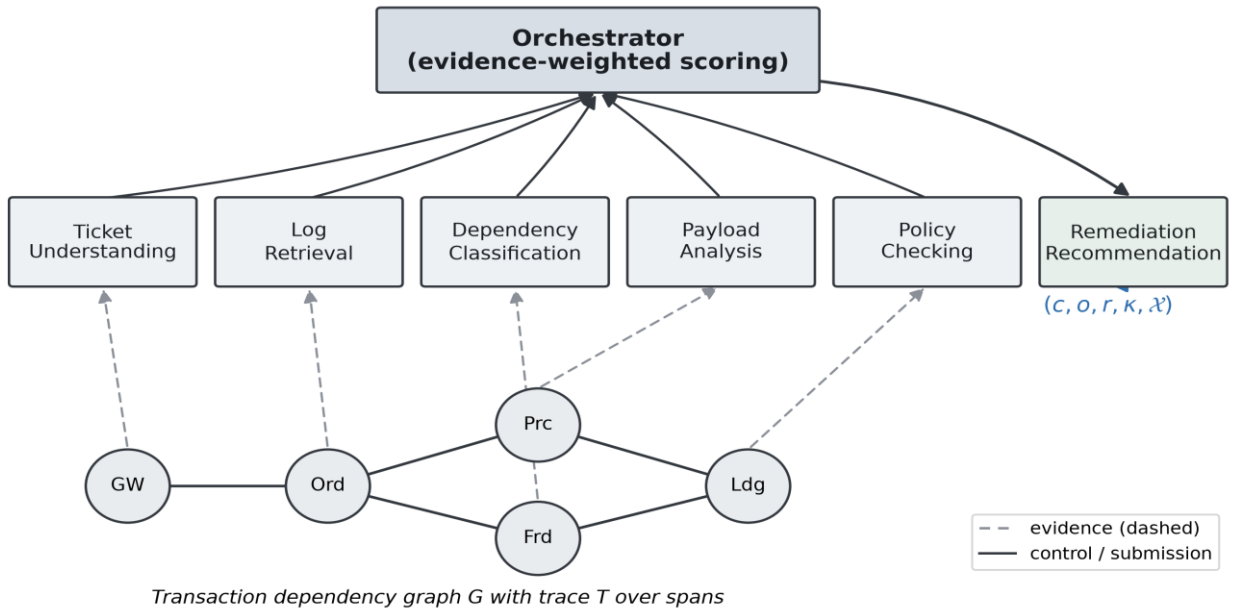

Fig 1. CARMA architecture. Six specialized agents draw evidence (dashed) from the trace and logs of a failed transaction over its dependency graph G, and submit scored evidence items to the orchestrator, which computes an evidence-weighted root-

cause score and invokes the remediation agent to produce the final (c, o, r, κ, X).

5.2 Agents

Each agent below is specified by its responsibility, inputs, outputs, failure modes, and controls.

Ticket-Understanding Agent. Responsibility: parse the incident ticket Q into a structured query—affected operation, edge symptom, time window, any user-supplied identifiers—and extract the trace id when present. Inputs: Q. Outputs: a structured intent record and a prior over plausible classes. Failure modes: over-reading a terse ticket and inventing detail; anchoring later agents on a wrong symptom. Controls: the output is constrained to fields extractable from Q, and any inferred field is flagged low-confidence so it cannot dominate aggregation.

Log-Retrieval Agent. Responsibility: fetch and rank log records correlated to the trace id across all services on the path, and surface the earliest error-bearing line per service. Inputs: trace id, G, log stores. Outputs: ranked log evidence with source service and timestamp. Failure modes: retrieval recall gaps when logs are sampled or clock-skewed; returning the loudest rather than the earliest error. Controls: retrieval is grounded in the trace-context identifier [6], results are ordered by corrected timestamp, and the agent reports retrieval coverage so the orchestrator can discount low-coverage evidence.

Dependency-Classification Agent. Responsibility: examine the trace T over G to classify each failing edge as upstream-caused, internal, or downstream-caused, and identify the originating hop. Inputs: T, G, edge timeout/retry configuration. Outputs: per-edge causal direction and a candidate originating service o. Failure modes: mistaking timeout propagation for the originating fault; missing a fault on a sampled or missing span. Controls: it applies span timing against declared timeout and retry budgets [10] so that a tripped circuit breaker is recognized as a propagated symptom rather than a cause.

Payload-Analysis Agent. Responsibility: inspect request and response payloads on the failing edge for malformation, schema drift, or type mismatch. Inputs: captured payloads, declared schemas. Outputs: a structured diff and a malformation/drift verdict with the offending field. Failure modes: false drift signals from benign optional-field changes; missing payloads when capture is disabled. Controls: it compares against the declared schema version and emits evidence only when a concrete field-level discrepancy exists.

Policy-Checking Agent. Responsibility: determine whether the failure is an authorization mismatch or a business-rule rejection by checking the request against applicable policy and rule definitions. Inputs: request context, identity claims, policy/rule store. Outputs: the matched (or violated) clause and a policy-cause verdict. Failure modes: stale policy snapshots; conflating a legitimate denial with a defect. Controls: it cites the specific clause id, and a denial is only reported as a root cause when the trace shows the request was otherwise well-formed.

Remediation-Recommendation Agent. Responsibility: given the resolved class c and owner o, propose a scoped remediation r and the responsible team. Inputs: resolved (c, o), evidence set X, runbook corpus. Outputs: remediation text bounded to the evidence. Failure modes: recommending an action broader than the evidence justifies. Controls: remediation is generated after aggregation and conditioned on X, so it inherits the grounding of the resolved cause rather than proposing speculative fixes.

Orchestrator. Responsibility: sequence the agents, collect their scored evidence, compute the evidence-weighted root-cause score, decide whether to auto-resolve or escalate, and assemble the final output. Controls: it enforces that no class may win without at least one supporting evidence item, and it routes to a human whenever the top score fails to clear the threshold τ or two classes tie within a margin.

5.3 Failure-Mode Taxonomy and Signal Mapping

CARMA targets the seven failure classes in Table 2. The table is the framework's original contribution to legibility: it records, per class, which agent is expected to produce the decisive signal and what that signal looks like. This mapping is what makes the framework falsifiable—if the dependency agent never contributes the decisive signal for timeout propagation, the design is wrong, not merely underperforming.

Table 2. Failure-mode taxonomy and the per-agent signal mapping. "Decisive agent" is the agent expected to contribute the highest-weight evidence for that class; "decisive signal" is the concrete observable it emits.

Failure class	Decisive agent	Decisive signal	Supporting agents
Malformed request	Payload-Analysis	Field-level schema violation in the request payload on the entry edge	Ticket, Log
Authorization mismatch	Policy-Checking	Denied identity claim against a specific authorization clause	Log, Dependency
Cache inconsistency	Log-Retrieval	Stale-read log line; response passes structurally but fails a later consistency check	Payload, Dependency
Timeout propagation	Dependency-Classification	Span latency exceeds the configured timeout/retry budget; breaker trips downstream of the slow hop	Log
Dependency unavailability	Dependency-Classification	Downstream span returns connection-refused / 5xx as the earliest error on the path	Log
Schema drift	Payload-Analysis	Declared schema version differs from observed response shape; deserialization fails	Log, Dependency
Business-rule rejection	Policy-Checking	Violated business-rule clause on an otherwise well-formed request	Ticket, Payload

5.4 Evidence-Weighted Aggregation

The orchestrator's scoring rule is the framework's original analytical angle. Let A be the set of agents and C the root-cause classes. Each agent a ∈ A emits zero or more evidence items; let e_{a,c} ∈ [0,1] be agent a's confidence that the cause is class c (zero if it emits nothing for c). Each agent's contribution is weighted by two factors: a fixed competence weight w_{a,c} encoding the prior that agent a is authoritative for class c (read directly off Table 2—the decisive agent gets the largest weight), and a coverage factor γ_a ∈ [0,1] that the agent self-

reports, discounting evidence drawn from incomplete retrieval. The score for class c is

$$S(c) = [\ \Sigma_a\ w_{a,c} \cdot \gamma_a \cdot e_{a,c}\ ] / [\ \Sigma_a\ w_{a,c}\ ] \quad (1)$$

The framework selects $\hat{c} = \text{argmax}_c\ S(c)$ and sets the confidence $\kappa = S(\hat{c})$. The owner $\hat{o}$ is taken from the dependency agent's originating-hop evidence associated with $\hat{c}$. The orchestrator auto-resolves only if $\kappa \geq \tau$ and the margin $S(\hat{c}) - S(c_2)$ over the runner-up c_2 exceeds a separation threshold δ; otherwise it escalates with the full evidence set attached. Two properties follow by construction. First, grounding: if no agent emits evidence for c, then $S(c) = 0$, so a class with no supporting signal can never be selected. Second, robustness to a single miscalibrated agent: because weights are normalized and coverage-discounted, one overconfident agent cannot by itself clear τ unless it is the decisive agent for that class and its evidence is well-covered.

Algorithm 1. CARMA orchestration for one diagnosis episode.

```
input (T, L, Q, G); thresholds τ, δ
intent ← TicketUnderstanding(Q)
logs ← LogRetrieval(intent.traceId, G, L)
evidence ← ∅
for all a ∈ {Dependency, Payload, Policy, Log}:
    evidence ← evidence ∪ a(T, G, logs, intent)
for all c ∈ C:  compute S(c) from evidence via weighted rule
ĉ ← argmax_c S(c);  κ ← S(ĉ)
ô ← originating hop for ĉ (Dependency agent)
if κ ≥ τ and S(ĉ) − S(c₂) > δ:
    r ← Remediation(ĉ, ô, X);  return auto-resolve (ĉ, ô, r, κ, X)
else:  return escalate with (X, ranked S)
```

6. EVALUATION DESIGN

Because a labeled corpus of real cross-team incidents is not available, CARMA is evaluated on a synthetic benchmark whose ground truth is known by construction. This explicitly trades realism for labeled control; Section 9 treats that trade-off as the primary threat to validity.

6.1 Benchmark Generator

The generator produces episodes in three stages. (1) Topology. It samples a dependency graph G of between five and twelve services with a single edge-facing entry point, assigns each service an owner, and annotates each edge with a timeout and a retry budget drawn from realistic ranges. (2) Nominal transaction. It simulates a successful transaction over G, emitting a trace T of spans with plausible per-hop latencies and a correlated log bundle L keyed by a generated trace id, following the W3C propagation model [6]. (3) Fault injection. It selects one of the seven classes in Table 2 and a target hop, then mutates the nominal artifacts to realize that fault: a malformed request mutates a payload field; timeout propagation inflates a downstream span past its budget and trips the upstream breaker; schema drift bumps a declared schema version without updating the deserializer; a business-rule rejection injects a violated clause; and so on. The ground-truth triple (c*, o*, r*) is recorded. A terse ticket Q is synthesized from the edge symptom only, mirroring the information-poor tickets engineers actually receive. The generator is seeded, so episode sets are reproducible, and class balance is configurable.

6.2 Metrics

Seven metrics are reported. Root-cause classification accuracy is the fraction of episodes with $\hat{c} = c^*$; joint class-and-owner accuracy is also reported. MTTD (mean-time-to-diagnosis) is the wall-clock time from episode start to returned diagnosis. False escalation rate is the fraction of episodes escalated to a human despite the framework holding sufficient evidence (ground truth recoverable from the emitted evidence set). Remediation precision is the fraction of auto-resolved episodes whose recommended r matches the recorded r* class. Human-review workload is the fraction of episodes requiring human attention, the operational quantity the framework aims to reduce. Latency and cost are the per-episode end-to-end time and the aggregate model-token usage across agents, included because the multi-agent design trades both against accuracy.

6.3 Baselines

The comparison is against three baselines. Rule-based triage encodes threshold-and-string rules and attributes to the alarming service, the production status quo. Single-agent LLM receives the full (T, L, Q, G) in one prompt and is asked for (c, o, r) in one pass, with no agent decomposition. Retrieval-only returns the top class implied by the highest-ranked retrieved log line, with no cross-signal aggregation. These isolate the contributions of decomposition (single-agent vs. CARMA) and of aggregation (retrieval-only vs. CARMA).

7. RESULTS AND ILLUSTRATIVE ANALYSIS

All numbers in this section are illustrative figures produced by the CARMA simulation harness on seeded synthetic episodes. They are intended to characterize the framework's qualitative behavior and the trade-offs it exposes; they are not measurements from a production incident corpus and should not be read as such.

Table 3 reports the headline comparison over a balanced synthetic set across the seven fault classes. CARMA's evidence-weighted aggregation yields the highest illustrative classification accuracy and the lowest human-review workload, at the cost of higher latency and token cost than the single-pass baselines, which is the expected price of running six agents. The single-agent LLM matches CARMA on simple single-signal faults but loses ground on faults that require fusing the trace with a payload diff or a policy clause, and its false escalation behavior is erratic because it has no calibrated confidence to gate on. Rule-based triage is fastest and cheapest but attributes to the alarming rather than the originating service on propagated faults, which inflates its effective workload once mis-routes are counted.

Table 3. Illustrative end-to-end comparison on the synthetic benchmark (balanced over seven fault classes). Values are simulation-harness outputs, not production measurements. Latency and cost are normalized to the rule-based baseline (1.0×).

Method	Class acc.	Class+owner acc.	MTTD (s)	False escal.	Remed. prec.	Human workload	Lat./Cost
Rule-based triage	0.46	0.39	4	0.07	0.41	0.71	1.0× / 1.0×
Retrieval-only	0.58	0.44	9	0.22	0.49	0.55	1.4× / 2.1×
Single-agent LLM	0.74	0.63	14	0.19	0.68	0.38	2.0× / 4.6×
CARMA (this work)	0.88	0.82	21	0.09	0.83	0.21	3.3× / 7.9×

Fig 2. Illustrative (simulated) end-to-end comparison. CARMA leads on class and class+owner accuracy and on remediation precision, and carries the lowest human-review workload, across the four methods.

The numbers make the trade-off concrete. CARMA lifts class accuracy from the single-agent baseline's 0.74 to 0.88 (+0.14), but the joint class-and-owner accuracy gap is wider still, 0.63 to 0.82 (+0.19), which matters operationally because attributing a fault to the wrong team is a mis-route that costs a full hand-off. Human-review workload falls monotonically along the capability axis—0.71 for rule-based, 0.55 retrieval-only, 0.38 single-agent, 0.21 for CARMA—so CARMA leaves roughly a third of the residual human effort of the single-agent assistant and under a third of the rule-based baseline. This comes at a real price: 3.3× latency and 7.9× token cost over rule-based triage, the near-linear cost of invoking six agents. The MTTD of twenty-one seconds, however, must be read against human diagnosis time measured in tens of minutes, so the relevant comparison is agent-versus-human, not agent-versus-rule.

Figure 3 breaks accuracy down by fault class for the two strongest methods. The pattern is informative: the single-agent baseline is competitive on malformed-request and dependency-unavailability faults, where a single signal is decisive, but trails CARMA most on cache inconsistency and schema drift, where the decisive signal must be combined with a corroborating one. Quantitatively, the margin is smallest on dependency-unavailability (0.91 vs. 0.88, a 0.03 gap—one signal suffices) and largest on schema drift (0.85 vs. 0.69, 0.16) and cache inconsistency (0.86 vs. 0.71, 0.15). This is consistent with the framework's premise—the value of decomposition concentrates on faults that require cross-signal fusion, not on faults already legible from one signal.

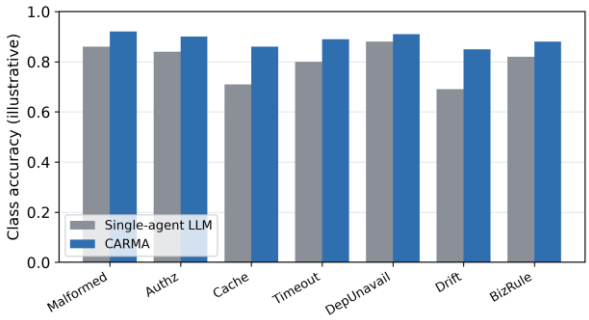

Fig 3. Illustrative per-fault-class accuracy. CARMA's margin is largest on cache inconsistency and schema drift, the classes whose decisive signal must be fused with a corroborating one. Simulated values.

Table 4 reports an ablation that removes one agent at a time. Removing the dependency-classification agent collapses accuracy on timeout propagation and dependency unavailability, since those classes lose their decisive agent and the orchestrator can no longer separate a propagated symptom from the originating hop; owner accuracy falls sharply. Removing the policy-checking agent leaves the structural classes largely intact but degrades authorization-mismatch and business-rule classes, and notably raises false escalation because the orchestrator can no longer reach τ on policy-driven faults and defers them to humans. The ablation supports the signal-mapping in Table 2: each agent's removal degrades precisely the classes for which it is the decisive agent.

Table 4. Illustrative ablation. Removing an agent degrades the classes for which it is the decisive agent (cf. Table 2). Simulated values.

Configuration	Class acc.	Owner acc.	False escal.
Full CARMA	0.88	0.85	0.09
− Dependency agent	0.71	0.58	0.14
− Policy agent	0.79	0.81	0.17
− Payload agent	0.76	0.80	0.13

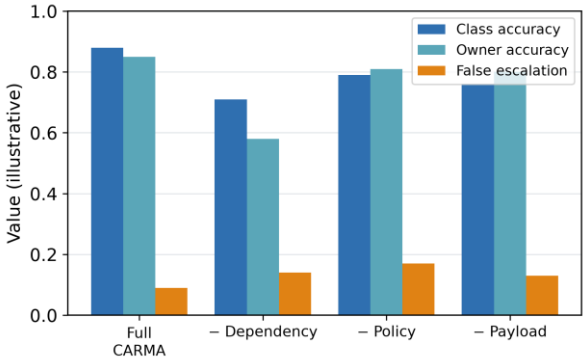

Fig 4. Illustrative (simulated) ablation. Each agent's removal degrades class accuracy, owner accuracy, or false-escalation rate on precisely the classes for which it is the decisive agent.

Reading the ablation closely reinforces the joint class-and-owner objective. Removing the dependency agent costs 0.17 of class accuracy but 0.27 of owner accuracy (0.85 to 0.58)—a much larger relative fall—because the originating hop ô is drawn from that agent's evidence, so without it the framework can still sometimes name the fault type but not the team that owns it. The policy agent's removal barely moves class accuracy on structural faults yet nearly doubles false escalation (0.09 to 0.17), the signature of an orchestrator that has lost its decisive signal for a class and correctly refuses to auto-resolve rather than guess. The ablation therefore validates two design choices at once: the per-class signal mapping and the grounding rule that prefers escalation over an unsupported verdict.

8. DISCUSSION

The results, illustrative as they are, sharpen a trade-off that any team adopting an agentic diagnosis system must confront. Adding agents buys attribution quality on cross-signal faults and lowers human workload, but it spends latency and token cost roughly linearly in the number of agents invoked. In the illustrative harness, CARMA's MTTD of twenty-one seconds is longer than a single-pass model's, yet it is still far below the tens of minutes a human spends stitching logs, so the relevant comparison is not agent-versus-agent latency but agent-versus-human diagnosis time. The cost multiplier is real, however, and argues for invoking the full agent set only when cheap baselines abstain—a cascade in which rule-based triage handles recurring faults and CARMA is reserved for the novel, cross-boundary incidents where its fusion pays off.

A second trade-off is automation against false escalation. The auto-resolution threshold τ is a dial between two failure modes: setting it high suppresses misclassification but escalates more incidents to humans, eroding the workload savings; setting it low resolves more incidents automatically but risks acting on a weak diagnosis. The evidence-weighted rule helps here precisely because κ is grounded in covered evidence rather than model self-assurance, so τ gates on something more meaningful than a single model's stated confidence. The right operating point is organizational: a team where a wrong automated remediation is cheap to reverse can run a lower τ than one where it is not.

9. THREATS TO VALIDITY AND LIMITATIONS

The dominant threat is synthetic realism. The generator injects faults chosen to model and produces telemetry cleaner than production—real logs are noisier, traces are sampled and incomplete, clocks skew further, and faults co-occur. A framework tuned on synthetic episodes may overfit to the generator's regularities, and the reported numbers would very likely soften on real incidents. There is no real labeled incident corpus, so external validity cannot be claimed; the contribution is the framework design and a reproducible protocol, not a production benchmark. Generalization is a second concern: the seven classes are representative, not exhaustive, and real incidents are often combinations—a schema drift that causes a timeout that trips a breaker—which strain a single-class output and suggest a multi-label extension. Third, residual hallucination is bounded but not eliminated: evidence-weighted aggregation guarantees a selected class has some supporting evidence, but it does not guarantee the evidence is correctly interpreted, and the remediation agent can still over-generalize within the bounds of X. Finally, the competence weights w_{a,c} are set from the taxonomy by hand; learning them from labeled outcomes is future work and would itself require the real corpus that is lacking.

10. PRACTICAL IMPLICATIONS

For SRE and on-call teams, the immediate implication is structural rather than algorithmic: instrument transactions so that the trace id propagates across every boundary and logs are queryable by it, because every agent in CARMA—and every human doing the same job—depends on that correlation being cheap [6], [7]. The quality of that telemetry itself bounds what any diagnostic agent can conclude, which motivates treating telemetry quality as an explicit input-readiness signal rather than an assumption [19]. Teams should treat attribution and ownership as a first-class output, since mis-routing to the alarming rather than the originating team is a large and often uncounted cost. Platform teams considering an agentic assistant should deploy it as a cascade behind existing rule-based triage rather than as a replacement, reserve the expensive multi-agent path for novel cross-boundary incidents, and keep a human in the loop above a calibrated confidence threshold, consistent with risk-management guidance for AI in consequential operations [17]. Most importantly, any such system should be required to cite its evidence: a diagnosis an engineer cannot audit is a liability during an incident, not an asset.

11. CONCLUSION

This paper presented CARMA, a context-aware multi-agent framework for diagnosing distributed transaction failures that decomposes the diagnostic task across six specialized agents and an orchestrator, and resolves a root cause through evidence-weighted aggregation that keeps the output grounded in retrieved signals. The framework's specific contributions are the agent decomposition with explicit failure modes and controls, an original per-fault-class signal mapping that makes the framework's reasoning legible and its evaluation falsifiable, and a reproducible synthetic benchmark and protocol over seven representative failure classes. Illustrative results from the simulation harness suggest that the value of decomposition concentrates on cross-signal faults and that the framework's accuracy and workload gains come at a predictable latency and cost premium. Future work is threefold: replace hand-set competence weights with weights learned from a labeled incident corpus; extend the output to multi-label causes for co-occurring faults; and validate the framework against real cross-team incidents to test whether the synthetic gains survive contact with production noise.

12. ACKNOWLEDGMENTS AND AI DISCLOSURE

Generative AI tools were used to assist with outline refinement, drafting support, and editorial review. The author verified all technical claims, citations, figures, and conclusions and is fully responsible for the content of this manuscript.

13. REFERENCES

[1] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In Int. Conf. on Learning Representations (ICLR).

[2] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Advances in Neural Information Processing Systems (NeurIPS).

[3] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Advances in Neural Information Processing Systems (NeurIPS).

[4] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

[5] Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., and Shanbhag, C. 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Google Technical Report dapper-2010-1.

[6] World Wide Web Consortium (W3C). 2021. Trace Context. W3C Recommendation. https://www.w3.org/TR/trace-context/

[7] Cloud Native Computing Foundation. 2024. OpenTelemetry Specification. https://opentelemetry.io/docs/specs/otel/

[8] Beyer, B., Jones, C., Petoff, J., and Murphy, N. R. 2016. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media.

[9] International Organization for Standardization. 2011. ISO/IEC 25010:2011 Systems and Software Engineering—SQuaRE—System and Software Quality Models.

[10] Nygard, M. T. 2018. Release It! Design and Deploy Production-Ready Software, 2nd ed. Pragmatic

Bookshelf.

[11] Chen, P., Qi, Y., Zheng, P., and Hou, D. 2014. CauseInfer: Automatic and Distributed Performance Diagnosis with Hierarchical Causality Graph in Large Distributed Systems. IEEE INFOCOM, 1887–1895.

[12] Kasikci, B., Schubert, B., Pereira, C., Pokam, G., and Candea, G. 2015. Failure Sketching: A Technique for Automated Root Cause Diagnosis of In-Production Failures. In Proc. 25th Symposium on Operating Systems Principles (SOSP).

[13] Pearl, J. 2009. Causality: Models, Reasoning, and Inference, 2nd ed. Cambridge University Press.

[14] Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q., and He, C. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In Proc. ESEC/FSE.

[15] Notaro, P., Cardoso, J., and Gerndt, M. 2022. A Survey of AIOps for Failure Management in the Era of Large Language Models. ACM Computing Surveys 54, 11s.

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. 2017. Attention Is All You Need. In Advances in Neural Information Processing Systems (NeurIPS).

[17] National Institute of Standards and Technology. 2023. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. https://doi.org/10.6028/NIST.AI.100-1

[18] Divi, V. R. 2026. Multi-Agent Debate for Software Architecture Governance: A Framework for ADR Generation, Risk Review, and Deployment Readiness. International Journal of Engineering Development and Research (IJEDR) 14, 2, 862–. DOI: 10.56975/ijedr.v14i2.308420.

[19] Gullapalli, S. 2026. Telemetry Quality Indexing for AI-Ready Observability in Multi-Node Cloud Systems: A Multi-Dimensional Framework for Distributed Infrastructure Diagnosis. International Journal of Engineering Development and Research (IJEDR) 14, 2, 878–. DOI: 10.56975/ijedr.v14i2.308419.