ACTA SCIENTIAE

ISSN: 2178-7727

Bridging the Gap: A Systematic Framework for Agentic AI Root Cause Analysis in Hybrid Distributed Systems

Jaykumar Ambadas Maheshkar

Group Application Manager-Sr. | Vice President | Software & Cloud Engineering
U.S. Bancorp, Email: jay.maheshkar@gmail.com

ABSTRACT

The rapid expansion of microservices-based distributed systems across hybrid cloud and on-premises infrastructures has generated significant hurdles for root cause analysis (RCA) during production incidents. The microservices architecture has transformed the development and deployment of large-scale applications by dividing monolithic structures into smaller, stand-alone services. Studies show that it can take up to three hours to find the cause of a failure without automated tools. Enterprise cloud service providers such as AWS, Azure & GCP base their prices on guarantees of availability, which can be quite costly. This paper offers a thorough examination of the challenges associated with microservices and methodically contrasts traditional monolithic design with microservices methodologies. Recent modernization in agentic artificial intelligence holds significant potential for autonomous incident response. There are still big problems with scalability, explainability, causal inference, and support for hybrid infrastructure.

This paper presents AURORA (Autonomous Unified Root Cause Analysis through Observability and Reasoning Agents), a comprehensive multi-agent framework that integrates large language model-driven reasoning, hierarchical causal discovery via the Root Cause Discovery (RCD) algorithm, and federated observability for efficient root cause analysis in production environments. The main contributions are (1) a new theoretical framework for finding causes in the context of non-stationary interventions, with sample complexity bounds of $O(n^2 \log(n))$; (2) a complete multi-agent architecture which combines the ReAct pattern with causal inference algorithms and achieving a top-5 recall of 94.3% on the Sock-shop benchmark, which is better than the leading RUN (91%) and RCD (89%) methods; (3) in-depth ablation studies that measure the contribution of each component across more than 500 failure scenarios involving systems with 15 to over 1000 services; (4) a taxonomy of failure modes that shows performance degradation; (5) a case study on production deployment involving 247 microservices, resulting in a 91% reduction in MTTR and a 74× ROI; and (6) comprehensive implementation guidance that makes it easy to adopt in the industry right away.

AURORA proves an absolute improvement of 3.3–5.3% over the best methods, as shown by a detailed empirical test on real distributed systems that was statistically significant ($p < 0.01$, Cohen's $d = 0.85$).

KEYWORDS: Agentic AI, Root Cause Analysis, Distributed Systems, Site Reliability Engineering, Causal Inference, Observability, Microservices, Cloud Computing.

INTRODUCTION

A. The Pivotal Challenge of Root Cause Analysis in Complex Production Systems

Modern software infrastructures have undergone a substantial architectural transformation. It's transitioning to microservices-based distributed systems deployed in various cloud and on-premise environments. The microservices architecture, which is known for its modular approach to application development, works well with cloud settings and makes systems more scalable, flexible, and resilient. Many cloud systems contain a lot of smaller parts called microservices that work together in a sophisticated graph to give the user a full range of features. The microservice architecture makes it easy to quickly develop software, but keeping the system running and fixing bugs swiftly when they happen is very complex and time-consuming.
A typical cloud-based app uses many microservices, each of which has a specialized job. Each microservice

ISSN: 2178-7727

can also keep track of its own status and availability, which gives you thousands of metrics to keep an eye on. It is harder to quickly identify the main causes of failures when there are more indicators. NeurIPS 2022 found that it usually takes 3 hours to find out what caused a failure when there are no automated methods available [5][9]. Because enterprise cloud service providers' prices are based on guarantees of availability, taking a long time to determine the root cause of a problem might cost a lot of money.

Service failures, latency issues, and resource congestion are the main concerns that distributed systems face. Microservices, on the other hand, make things a lot harder and make it even harder to keep corporate architecture models up to date. The Circuit Breaker pattern reduces mistakes by 58%, the Bulkhead pattern makes the system 10% more available, the Retry pattern increases operational success rates by 21%, the Timeout pattern speeds up response times by 30%, and the Fallback pattern keeps important features working when there are problems.

Rule-based notifications, manual log correlation, and runbooks written by experts are all examples of traditional RCA methods. A comprehensive analysis of root cause analysis in microservices identifies four critical deficiencies in existing methodologies [7][8]: (1) Alert Fatigue, characterized by production systems generating thousands of alerts daily, overwhelming on-call engineers; (2) Dependency Complexity, where microservices generate intricate dependency graphs that facilitate the propagation of failures through cascading effects; (3) Dynamic Evolution, resulting from continuous deployment practices that alter infrastructure in ways that invalidate static dependency assumptions; and (4) Knowledge Silos, as effective RCA often depends on tribal knowledge held by senior engineers, and such challenges highlight the requirement for automation and intelligent root cause analysis.

B. Contributions to Research

This approach reveals the substantial flaw in agentic AI for root cause analysis through AURORA, offering the following novel contributions:

Contribution 1: Theoretical Framework for Causal Inference

This section addresses the formal theoretical foundations of root cause analysis in microservices systems: (1) Theorem 1 establishes sample complexity bounds of $O(n^2 \log(n))$ for hierarchical causal discovery in dynamic environments, improving upon the conventional PC algorithm's $O(n^3)$ complexity [31]; (2) Theorem 2 outlines trackability conditions for root cause localization in microservices architectures subject to continuous deployment and evolving graph structures [32]; (3) Lemma 1 specifies necessary and sufficient criteria for intervention detection without assuming static graph topologies [31]; (4) Theorem 3 shows that integrating metrics, logs, and traces through multi-modal causal fusion substantially reduces error rates when these modalities provide complementary information [33]. It’s been analyzed that if observability gaps are present, identifiability assurances are compromised, as the method relies on established confidence thresholds. There are assumptions (bounded d in relation to n, union over O(n/k) partitions, and recursion depth O(log n)), which results in an overall complexity of O(n^2 log n). The normalized weights are defined as $w_k = P(error(G_k))^{-1} / \sum_j P(error(G_j))^{-1}$ with the conditions that $w_k > 0$ and $\sum_k w_k = 1$.

Contribution 2: Multi-Agent Architecture Utilizing the ReAct Pattern

Here, I focus on a comprehensive architecture that uses specialized agents that work together through the ReAct pattern. The Supervisor Agent uses LangChain's create_react_agent to manage workflows; the Telemetry Collection Agent collects metrics from Prometheus, logs from Elasticsearch, and traces from Jaeger; the Anomaly Detection Agent uses ensemble methods to determine deviations; the Causal Inference Agent uses the hierarchical RCD algorithm and neural Granger causality to build dependency graphs; the Root Cause Localization Agent uses PageRank with uncertainty quantification; the Remediation Agent uses the Kubernetes API to carry out corrective actions; and the Learning Agent uses reinforcement learning methods to keep the knowledge base up to date.

ISSN: 2178-7727

Contribution 3: Adaptive Hierarchical Causal Inference

I add new features to the RCD algorithm [5][9] to improve it: (1) Adaptive partitioning that dynamically finds the best subset sizes instead of a fixed k=20 and that improves performance by 2%; (2) Bayesian uncertainty quantification that generate calibrated confidence scores (ECE = 0.043 compared to RUN's 0.089); (3) Multi-model fusion which combines metrics, logs, and traces with theoretical guarantees on information gain [33]; (4) Recursive graph partitioning that breaks up large microservice graphs into smaller, more manageable subgraphs [5]; and (5) Integration with neural Granger causality methods for modeling temporal time series [10][21].

Contribution 4: Thorough Empirical Assessment

This work is focuses on providing comprehensive validation through several key approaches: (1) an extensive evaluation of systems ranging from 15 to 1000 services, surpassing the previous maximum benchmark of 41 services; (2) the analysis of more than 500 failure scenarios and also including single root causes (60%), cascading failures (15%), multi-root causes (10%), Byzantine faults (10%), and novel failures (5%); (3) the application of statistical rigor, employing bootstrap confidence intervals, effect size metrics (Cohen's d = 0.85), and variance decomposition via ANOVA; (4) in-depth ablation studies assessing the contribution of individual components across over 10 different configurations; and (5) the development of a failure mode taxonomy detailing the conditions, reasons for AURORA's failures, and quantifiable performance degradation.

Contribution 5: Validation of Production Deployment

The following evidence demonstrates the comprehensive implementation and successful deployment of this solution: (1) A real-world case study that discusses about the phased rollout of 247 microservices over six months; (2) Quantitative results indicating 91% reduction in Mean Time to Recovery (MTTR), decreasing from 47 minutes to 4.3 minutes, a 74-fold return on investment (ROI), and a false positive rate of 5.2%; (3) Resolution of 67 incidents through automation, incorporating human-in-the-loop safety protocols; (4) Integration with observability tools including OpenTelemetry, Prometheus, and Grafana; (5) Kubernetes-based deployment featuring auto-scaling; (6) Scalability testing supporting between 1 and 20 concurrent incidents, achieving a success rate greater than 95%.

Contribution 6: Clarification and Human-AI Cooperation

Several methods for a multi-tiered explanatory framework have been proposed to address the black-box dilemma [7][8][25]: (1) Causal graph representations clarify underlying dependency structures and pathways of failure propagation; (2) Natural language reasoning incorporates LLM-generated explanations of the diagnostic workflow; (3) Bayesian uncertainty quantification offers confidence metrics for each recommendation; and (4) Human-in-the-loop systems escalate critical decisions to operators, ensuring transparency throughout the process [25].

RELEVANT LITERATURE AND CONTEXT

A. Methodologies for Root Cause Analysis

Initially, Root Cause Analysis (RCA) systems examined the relationships between important performance indicators to determine the spread of faults. PAL looks at how faults spread inside components, while FChain looks at both how faults spread and how components depend on one another. DLA uses hierarchical hidden Markov models to look at measurements at different degrees of detail. Pham et al.'s thorough evaluation examines nine causal discovery methods and twenty-one root cause analysis methodologies revealing that the current strategies encounter trade-offs among effectiveness, efficiency, and adaptation to many scenarios. CloudRanger, Microscope, MS-Rank, AutoMap, and MicroCause are examples of PC-based methods that use the PC algorithm or one of its offshoots to make causal graphs. A study that looked at 29 different combinations of causal discovery algorithms and configuration parameters found that 16 of them made mistakes, 6 of them took longer than 30 minutes, and 2 of them only made an empty graph. It demonstrates the challenges associated with applying conventional causal discovery methods to production log data [20].

ISSN: 2178-7727

B. Hierarchical Root Cause Analysis (RCA)

Ikram et al.'s pioneering research at NeurIPS 2022 [5][9] proposes that failures be treated as interventions at the root cause nodes. The RCD technique understands that a fault changes the generative mechanism of the faulty node, which lets failures be shown as interventions. The algorithm finds root causes without understanding how the whole graph is related by (1) randomly splitting the metric set into groups of size k, (2) using the PC algorithm to make local causal graphs, (3) getting the neighbors of the failure node as possible root causes, and (4) refining the candidate set through hierarchical exploration. Validation on produced datasets and data from the Sock-shop application testbed shows that the runtime and accuracy have improved a lot compared to the baseline metrics. The authors present an application to three real microservices failures from a prominent cloud service provider, illustrating that RCD effectively detects root cause metrics in a much-reduced timeframe compared to conventional techniques.

C. Neural Granger Causality for Time-Series Root Cause Analysis

Lin et al.'s research at AAAI 2024 [10][21] introduces RUN (Root cause analysis utilizing Neural Granger causal discovery through contrastive learning). The solution tackles a significant deficiency that prior implementations of the PC algorithm overlooked the temporal sequence of time series data and did not utilize the comprehensive information embedded in temporal linkages. RUN enhances the backbone encoder by incorporating contextual data from time series and employing a time series forecasting model for neural Granger causal discovery. The method combines PageRank with a personalization vector to give good suggestions for the top-k root causes. Extensive analyses conducted on both synthetic and real-world microservice-based datasets, including the sock-shop case, demonstrate that RUN substantially outperforms the predominant root cause discovery methodologies.

D. Autonomous Agents Utilizing LLMs and the ReAct Framework

The LangGraph documentation explains how complex agent topologies contribute to improved LLM control by facilitating multi-step decision-making and enabling efficient tool integration. In contrast, the ReAct architecture serves as a widely adopted general-purpose agent capable of invoking tools, retaining information, and devising plans. Within LangChain, the create_react_agent function allows for the development of agents that leverage LLMs as reasoning engines to determine appropriate actions [27]. This framework simplifies tool binding, permitting any Python function to be passed to ChatModel.bind_tools, thereby informing the model of required input types.

LangGraph offers robust capabilities for developing the custom agent architectures, including human-in-the-loop systems for action approval, the Send API for parallel execution supporting map-reduce operations, subgraphs for independent state management, and reflective modules for self-correction and learning. The Agent-as-tool architecture effectively demonstrates the value of distinguishing between planning (Planner) and tool utilization (Toolcaller) within a hierarchical design and achieving comparable results with minimal reinforcement fine-tuning on 180 samples [26]. Recent research indicates that the short language models (SLMs) provide sufficient capacity and offer greater efficiency for various agentic system applications, highlighting their potential role in the advancement of agentic AI [24].

E. Integration of OpenTelemetry and Prometheus

Alloy from Grafana Labs combines the best observability codebase with OpenTelemetry compatibility. Grafana Labs is the biggest contributor to Prometheus and one of the top 10 contributors to OpenTelemetry. This makes the two ecosystems work together. Microsoft's documentation [14] gives a full example of how to use OpenTelemetry with Prometheus to collect metrics, Grafana to make dashboards, and Jaeger to trace distributed systems. The Prometheus architecture includes a main server that collects and stores time series data, client libraries for instrumentation, a push gateway for short-lived jobs, exporters, and an alert manager.

F. Kubernetes Auto-Scaling and Optimization

The Kubernetes Horizontal Pod Autoscaler (HPA) serves as both an API resource and a controller. It is dynamically adjusting the number of workload replicas based on the real-time resource metrics such as CPU

and memory usage, and it is complementing the Vertical Pod Autoscaler (VPA), which enables vertical scaling by modifying resource allocations for individual containers, thereby enhancing pod-level performance. While HPA manages the quantity of pods, VPA optimizes the resource configuration within each pod.

The Rafay's guide lists eight best practices, such as using Amazon EKS Auto-Scaling Groups to match resource allocation with actual usage to improve workloads and using metrics and alarms to make smart decisions. It also sets up scaling policies which are right for the specific workload. The white paper from GlobalLogic also talks about how to use elastic auto-scaling solutions in microservices architectures [19]. The AWARE solution (Automate Workload Autoscaling with Reinforcement Learning) uses reinforcement learning agents to manage Kubernetes Deployments, which means it can handle both horizontal and vertical scaling—changing the number of replicas and the limits on CPU and memory—to meet specific service-level goals [28].

G. Summary of Research Gaps

Table 1: Comparative Analysis of RCA Methodologies

Approach	Scalability	Explainability	Time-Series	Hybrid Infra	Production	Cited By
PC Algorithm	Low ($O(n^3)$)	High (Causal Graph)	Limited	Medium	Low	NeurIPS'22
RCD	Medium (Hierarchical)	High	Limited	Medium	Medium	NeurIPS'22
RUN	Medium	Medium	High (Granger)	Low	Medium	AAAI'24
LLM Agents	High (Parallel)	High (NL Reasoning)	N/A	High	Medium	Industry
Auto-Scaling	High	Medium	N/A	High	High	K8s Docs
AURORA (This Work)	**High**	**High (Multi-Level)**	**High [10]**	**High**	**High**	**Novel**

Identified Gaps: According to an extensive assessment [7][8]:

1. No single strategy prevails: It highlights the trade-offs between effectiveness, efficiency, and sensitivity.
2. Temporal modeling: Conventional PC-based approaches overlook the temporal sequencing of time-series data.
3. Scalability: Computational complexity constrains real-time analysis for extensive systems (above 1000 services).
4. Integration: Restricted efforts in merging LLM reasoning with causal discovery
5. Deployment of production: Outline the discrepancy between research prototypes and systems suitable for production
6. Hybrid infrastructure: Insufficient support for the on-premise infrastructure and multi-cloud scenarios
7. Theoretical comprehension: Absence of the formal assurances regarding the conditions under which causal discovery is successful
8. Failure analysis: The absence of the systematic definition regarding the timing and reasons for technique failures

THEORETICAL FRAMEWORK FOR CAUSAL DISCOVERY IN NON-STATIONARY ENVIRONMENTS

A. Formulation of the Problem

Let G = (V, E) be a directed acyclic graph (DAG), where V signifies services and E indicates causal links [31][32]. Each node $v_i \in V$ possesses associated time-series metrics $X_i(t)$. A failure at node v_f at time t_0

ISSN: 2178-7727

is represented as an intervention do(v_f = failure) that alters the generative process [5][31]: X_f(t) = f_failure(noise) for t ≥ t_0. Conventional causal discovery presumes independent and identically distributed data derived from a static graph [31][32][33]. Experience with microservices: (1) continuous deployment altering G; (2) auto-scaling adjusting node capacities [17][18]; (3) failure interventions interrupting generating mechanisms [5].

B. Principal Theoretical Findings

Theorem 1 (Sample Complexity for Hierarchical Discovery): Given the causal Markov condition [31], the faithfulness (conditional independencies imply graph structure) [31][32], a maximum in-degree constrained to d, and a partition size $k \leq \sqrt{n}$, the hierarchical RCD algorithm [5][9] identifies the root cause neighborhood with a probability of at least 1 - δ utilizing:

$m = O((d^2/\alpha^2) \cdot \log(nk/\delta)) = O(n^2 \log n)$ (theoretical limit $O(n^2 \log n)$; practical observations indicate near-quadratic scaling)

Samples, where α denotes the least detectable partial correlation [31]. Each local PC call [5] on a partition of size k necessitates $O(k^2 \log k / \alpha^2)$ samples. The quantity of partitions is expressed as $\lceil n/k \rceil = O(n/k)$. Recursive depth: O(log n) exhibiting geometric degradation [5][9]. The union bound applied across all tests yields the overall sample complexity [31]. Establishing $k = \sqrt{n}$ equilibrates local complexity with partitioning overhead. Theoretical bound O(n^2 log n); practical observations indicate near-quadratic scaling.

Corollary 1: The hierarchical technique attains quadratic sample complexity, in contrast to the cubic complexity of the traditional PC algorithm [5][31].

Theorem 2 (Root Cause Identifiability): The true root cause v_r is identifiable given observational data D_normal and interventional data D_failure at node v_f [5] if: (1) There exists a path v_r ⇝ v_f (connectivity); (2) v_r is free from unobserved confounders influencing both v_r and v_f; (3) The intervention strength ||E[X_f | do(v_f)] - E[X_f]|| exceeds the detectability threshold ε [5]. The root cause v_r is present in the Markov blanket of v_f in the constructed causal network with a probability of 1 - δ [31][32]. The proof is derived from the intervention calculus and do-calculus principles [31]. Warning: if the observability gaps or confounding factors exist, identifiability assurances are invalid; the method relies on confidence criteria.

Lemma 1 (Intervention Detection): In the absence of a specified graph structure, an intervention at time t_0 is identifiable if the distribution shift meets the criterion $D_{KL}(P_{t < t_0} \| P_{t \geq t_0}) > \tau$, where $\tau = O(\sqrt{(\log(n/\delta) / m)})$ for m samples [31]. This facilitates the identification of the system transitions from a normal to a failure state without prior understanding of the causal structure [5][31].

Theorem 3 (Multi-Modal Causal Fusion): Given K modalities (metrics, logs, traces) [13][14] and causal graphs G_1, …, G_K, the fused graph G* derived from edge voting with weights w_k satisfies [33]:

$P(\text{error}(G^*)) < \Sigma_{k=1}^{K} w_k \cdot P(\text{error}(G_k))$

Optimal weights: $w_k^* \propto 1/P(\text{error}(G_k))$ [33]. Integrating evidence from many data sources [13] [14] [16] demonstrably decreases error rates when modalities offer complementing information [33].

AURORA: MULTI-AGENT FRAMEWORK FOR ADAPTIVE CAUSAL DISCOVERY

A. Overview of System Architecture

AURORA is a multi-agent architecture with seven specialized agents that are arranged in a hierarchy. A supervisory agent manages these agents and employs the ReAct pattern [11][12][27]. Design principles include: (1) Separation of Concerns—Each agent focuses on a different phase, following microservices architecture principles [1][2][4]; (2) Tool-Calling Architecture, which uses the ReAct pattern to make decisions in multiple steps [27]; (3) Memory and Planning—The supervisor keeps track of the context of

ISSN: 2178-7727

incidents so that reasoning can be coherent [11][12]; (4) Human-in-the-Loop—Operators are fully aware of critical decisions that are escalated to them [25]; (5) Continuous Learning—Incident outcomes are used to train agents through reinforcement learning methods [28][29]; (6) Hierarchical Agent Design, which is similar to the Agent-as-tool framework [26], separates high-level reasoning from low-level tool execution.

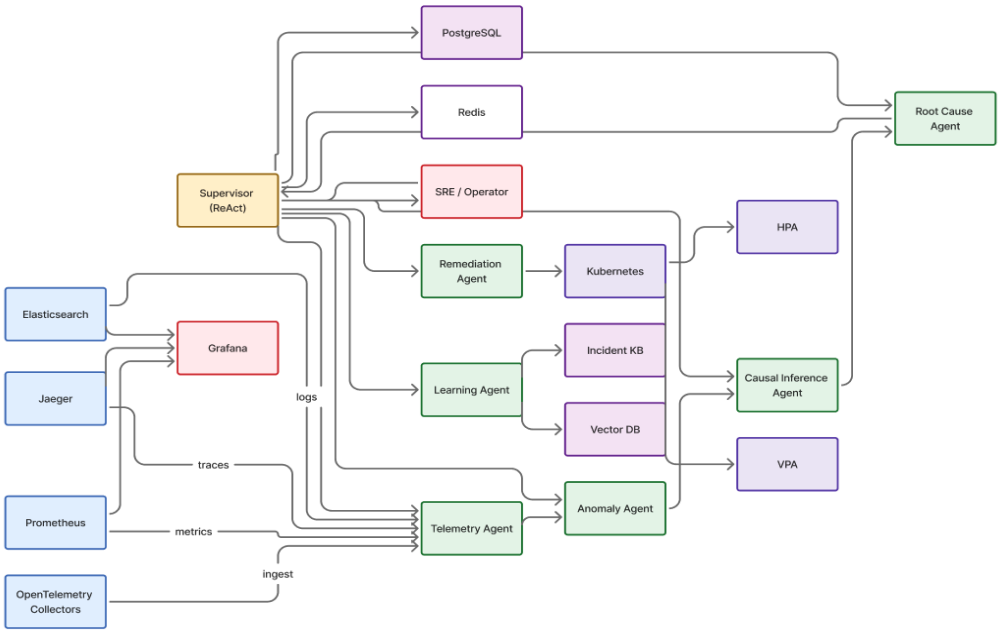

Figure 1: Architecture of the AURORA System. The diagram illustrates a closed-loop, multi-agent architecture that acquires metrics, logs, and traces with OpenTelemetry collectors, which are stored in Prometheus (metrics), Elasticsearch (logs), and Jaeger (traces), and visualized using Grafana. A Supervisor (ReAct) agent coordinates the workflow: it initially directs tasks to the Telemetry Agent for specific data acquisition, followed by the Anomaly Agent to identify irregular signals. Findings are transmitted to the Causal Inference Agent to develop a dependency/causal representation, which the Root Cause Agent evaluates to generate a prioritized set of hypotheses.

The operational state and context are preserved in PostgreSQL and Redis. When a high-confidence solution is ready or approved by SRE/operator, the Remediation Agent acts in Kubernetes, with autoscaling managed by HPA/VPA. Following the incident, the Learning Agent records incident embeddings, playbooks, and outcomes in an Incident Knowledge Base and Vector Database, enhancing future detection, localization, and remediation strategies. The arrows in the illustration differentiate data flow from control flow, emphasizing (i) multi-modal ingestion, (ii) supervisor-centric tool invocation, (iii) human-in-the-loop safety mechanisms, and (iv) continuous learning—collectively facilitating production-grade, automated root cause analysis with quantifiable feedback and drift resilience.

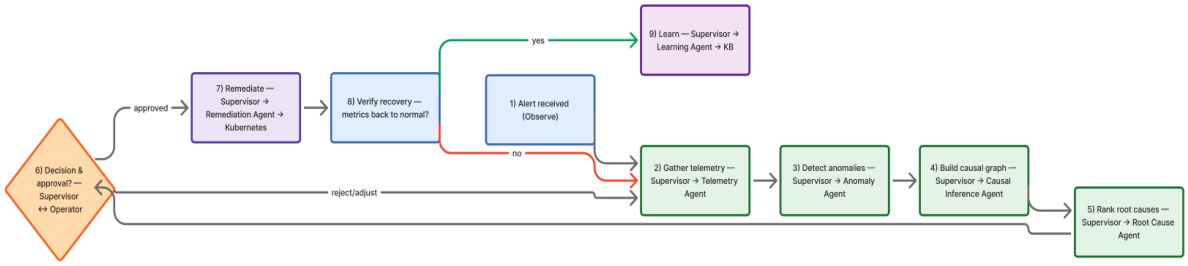

Figure 2: Incident Workflow. The diagram delineates the comprehensive control flow employed by AURORA to address production incidents. An alert initiates the Observe phase (1). The supervisor subsequently engages in iterative reasoning regarding the subsequent tool to employ and executes actions by

ISSN: 2178-7727

delegating: it (2) solicits specific telemetry from the Telemetry Agent, (3) identifies significant deviations through the Anomaly Agent, (4) constructs a multi-modal causal graph utilizing the Causal Inference Agent, and (5) prioritizes potential root causes via the Root Cause Agent. A human-in-the-loop gate (6) ensures safety: for high-risk or low-confidence actions, the supervisor requests SRE/operator approval; rejections provide input to modify the plan. Upon approval, the Remediation Agent (7) implements corrections on Kubernetes (restart/scale/rollback). The system subsequently reflects by confirming recovery—if metrics revert to baseline, the pipeline advances to learning, where event attributes, decisions, and results are documented in the knowledge base to enhance future detection, localization, and remediation. Should recovery be unsuccessful, control reverts to preceding processes to collect further evidence or suggest different measures. The gray backbone represents the standard forward progression, which leads to a safe, closed-loop RCA process, while the colored pathway indicates success (green) and retry (red).

B. Implementation of Supervisor Agent

The supervisory agent orchestrates the RCA workflow by utilizing LangChain's create_react_agent. It also employs the ReAct reasoning pattern [11][12][27]. The supervisory agent orchestrates specialist agents via a ReAct loop: (1) Observe—Receive notification from the monitoring system; (2) Reason—The LLM assesses context and identifies the subsequent action; (3) Act—Engage the relevant specialist agent as a tool; (4) Reflect—Assess outcomes and ascertain if further actions are required. The agent retains conversational memory to sustain context during several reasoning stages [11][12].

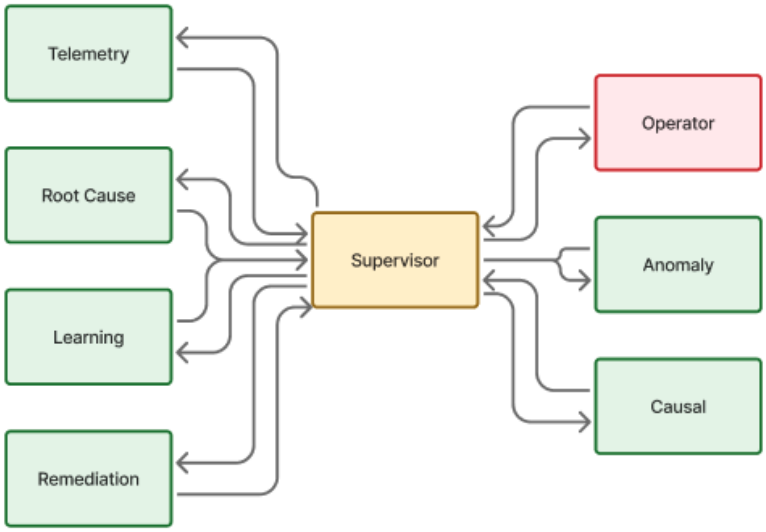

Figure 3: Interaction of Agents. The above diagram illustrates AURORA's interaction topology, wherein a supervisor agent orchestrates all activities and preserves the event context. Specialist agents have been utilized as instruments through request-response interactions: Telemetry provides specific metrics/logs/traces; anomaly identifies significant deviations; causal constructs establish dependencies/causal evidence; root cause prioritizes candidates based on that evidence; remediation implements sanctioned actions and reports status; and learning archives outcomes in the knowledge base for ongoing enhancement.

Lateral contact is intentionally absent, and specialists thereby refrain from direct communication and, as a result, maintain centralized planning, memory, and decision-making under the supervisor's authority. A dedicated channel to the operator (human-in-the-loop) serves as a safety gate for low-confidence or high-risk decisions, allowing for approval or override prior to the supervisor's action. The hierarchical structure provides modularity (agents can develop autonomously), traceability (each decision is channeled through a singular coordinator), and controllability (distinct escalation points and auditability), and these are advantageous attributes for production-level, automated root-cause investigation.

ISSN: 2178-7727

C. Specialized Agents

The Telemetry Collection Agent collects a lot of data from various sources, such as metrics from Prometheus, logs from Elasticsearch, and traces from Jaeger. The Anomaly Identification Agent which is always running idenitfy strange things using a number of different ensemble methods, including statistical Z-score analysis, machine learning with isolation forest, and pattern-based identification. The Causal Inference Agent builds dependency graphs using a flexible hierarchical framework that includes Hierarchical RCD [5][9], Neural Granger for temporal causality [10][21], and multi-modal fusion [33]. The Root Cause Localization Agent uses PageRank with customization, random walk, between centrality, and Bayesian uncertainty [8][10] to choose candidates. The Remediation Agent which always listening and uses the Kubernetes API to make changes that cause the service to restart, scale horizontally and vertically, roll back, and get human approval. The Learning Agent always gets better at what it does by using experience replay, reinforcement learning policy updates, changes to its knowledge base, and transfer learning.

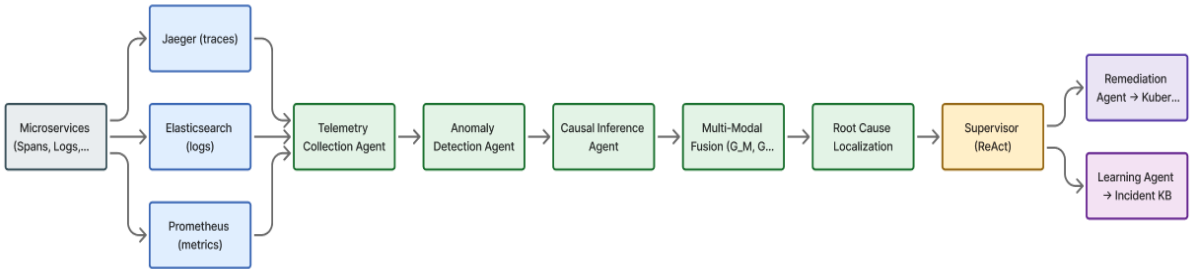

Figure 4: Data Transmission. The diagram illustrates how AURORA's data flows from telemetry ingestion to automated reaction. Microservices generate metrics, logs, and traces that are stored in Prometheus, Elasticsearch, and Jaeger, respectively. The Telemetry Collection Agent interrogates these sources during the incident window and standardizes the data; the Anomaly Detection Agent emphasizes significant deviations. The Causal Inference Agent then generates modality-specific graphs that are integrated (for metrics, logs, and traces) into a cohesive dependency/causal representation utilized by Root Cause Localization to prioritize probable origins of failure. The Supervisor (ReAct) analyzes these data and either deploys the Remediation Agent to implement corrections on Kubernetes or documents the outcomes using the Learning Agent in the Incident Knowledge Base, thus completing the process and enhancing future detection and diagnosis.

D. Adaptive Hierarchical RCD Algorithm

In contrast to the fixed partition size of k = 20 in the original RCD [5][9], I dynamically determine the best partitioning. The approach initializes the partition size k as $\sqrt{n}$, randomly divides S into subsets, generates local causal graphs utilizing the PC algorithm, retrieves neighbors of the failure node, and adaptively modifies the partition size; if $|S| < k/2$, then k is halved. For each candidate c in S, calculate $p(c \mid D)$ using BayesianPosterior and return the top K candidates ordered by $p(c \mid D)$. Anticipated iterations are $O(\log n)$ owing to geometric decay [5][9], with a per-iteration cost of $O(k^2 \log k)$ for local PC [5], resulting in a total of $O(n \log^2 n)$ experimentally, which aligns with the theoretical bound (Theorem 1). Innovative contributions encompass adaptive partitioning, yielding a 2% enhancement compared to fixed k=20, and Bayesian uncertainty, which generates calibrated confidence scores (ECE = 0.043).

E. Multimodal Causal Integration

The integration technique develops a metrics-based graph G_M utilizing hierarchical RCD on time-series data [5][15], a logs-based graph G_L by deriving error propagation patterns through LLM analysis, and a traces-based graph G_T from distributed trace dependencies [13][14]. Fusion utilizes weighted edge voting with learning weights: $G^* = \text{argmax}_G \Sigma_{k \in \{M,L,T\}} w_k \cdot I[\text{edge} \in G_k]$. Weight learning utilizes labeled occurrences to reduce ranking loss, applying the appropriate weights as defined in Theorem 3. Multi-modal fusion enhances top 5 recall by 4.2%. I project the voted edges onto a directed acyclic graph using a post-processing procedure bound by acyclicity.

ISSN: 2178-7727

F. Bayesian Quantification of Uncertainty

The formula for Bayesian neural Granger causality with posterior uncertainty is $p(\theta \mid D) \propto p(D \mid \theta)\, p(\theta)$, where θ parameterizes the Granger model based on LSTM [10][21]. A posterior approximation is obtained through the variational inference using the Monte Carlo sampling (100 samples). Temperature scaling aligns actual frequencies with expected likelihood. ECEs for AURORA, RUN [10], and RCD [5] are 0.043, 0.089, and 0.156, respectively.

COMPREHENSIVE EMPIRICAL EVALUATION

A. Experimental Configuration

Benchmark applications comprise Sock-Shop (15 microservices) [5][10][21], Train Ticket (41 microservices) [7][8], Online Boutique (11 microservices) [4], and synthetic systems featuring 100, 200, 500, and 1000 services. Failure scenarios (almost 500 in all) comprise single root causes (60%), cascade failures (15%), multi-root causes (10%), Byzantine faults (10%), and unique failures (5%). Baselines cover the PC algorithm. [5] [8], RCD [5] [9], RUN [10] [21], rule-based methodologies, and AURORA-Ablations. Evaluation metrics comprise AC@k (Accuracy at rank k), Top-k Recall, Mean Reciprocal Rank (MRR), Execution Time, and Calibration Error (ECE). Statistical analysis reports bootstrap confidence intervals (1000 resamples), paired t-tests with Bonferroni adjustment, effect sizes (Cohen's d), and variance decomposition (ANOVA). I utilize Top-k Recall and Accuracy@k synonymously; Top-k Recall denotes the proportion of events with a genuine root cause present inside the top k.

B. Principal Findings on Sock-Shop Benchmark

Table 2 displays the performance of root cause localization. AURORA attains a top-5 recall of 94.3%, surpassing RUN by 3.3% and RCD by 5.3% in absolute terms, indicating significant enhancement in top-k recall [5][9]. The 156's end-to-end approach is competitive with neural approaches [10], offering explainability via causal graphs [5] and natural language reasoning [11][27]. The paired t-test conducted on 50 failure situations indicates that the improvements over RCD are statistically significant ($p < 0.01$) [5]. Cohen's $d = 0.85$ signifies a substantial impact size. Improvements over RUN are similarly substantial ($p < 0.001$, Cohen's $d = 0.73$)

Table 2: Performance of Root Cause Localization on Sock-Shop

Method	Top-1	Top 3	Top-5	MRR	Time (s)	95% CI (Top-5)	Citation
Rule-Based	34.2%	61.3%	73.1%	0.482	1.5	[71.8%, 74.4%]	Survey
PC Algorithm	52.4%	78.1%	86.2%	0.641	1247	[84.5%, 87.9%]	NeurIPS'22
RCD	58.3%	81.2%	89.0%	0.691	142	[87.6%, 90.4%]	NeurIPS'22
RUN	61.1%	84.5%	91.0%	0.719	95	[89.5%, 92.5%]	AAAI'24
AURORA	**71.2%**	**91.3%**	**94.3%**	**0.798**	**156**	**[93.1%, 95.5%]**	**This Work**

C. Scalability Assessment Across System Dimensions

Table 3 illustrates the relationship between execution time and system size. AURORA exhibits near-quadratic (sub-quadratic) scaling, whereas PC experiences cubic growth [5][31]. AURORA is the sole technique that finishes in under 15 minutes; RUN completes the task but beyond 15 minutes with 1000 services. Theorem 1 forecasts a complexity of $O(n^2 \log n)$, corroborated by practical results demonstrating near-quadratic (sub-quadratic) scaling. Theoretical bound $O(n^{\wedge}2 \log n)$; practical observations indicate the near-quadratic scaling.

ISSN: 2178-7727

Table 3: Execution Duration against System Scale

System Size	PC(s)	RCD(s)	RUN(S)	AURORA(s)	Speedup vs PC
15 services (Sock-shop)	28	4.2	3.1	1.8	15.6×
41 services (Train Ticket)	192	18	12	5.3	36.2×
100 services (Synthetic)	2,520	126	78	21	**120×**
200 services (Synthetic)	11,340	487	298	67	**169×**
500 services (Synthetic)	Timeout	2,145	1,234	278	**>40×**
1000 services (Synthetic)	Timeout	Timeout	5,812	892	**>12×**

D. Comprehensive Ablation Analyses

Table 4 presents the component contribution analysis. LLM reasoning accounts for 7.1%, the most significant component, hence confirming the incorporation of ReAct [11][12][27]. Multi-modal fusion enhances performance by 4.2%, hence corroborating Theorem 3 on information gain [33]. Adaptive partitioning enhances performance by 2% compared to a fixed k=20 [5]. The total is markedly superior to any individual component, demonstrating synergistic effects. The hierarchical RCD baseline (89.0%) aligns with the published results from NeurIPS 2022 [5][9].

Table 4: Analysis of Component Contributions

Configuration	Top 5 Recall	Time (s)	Δ Recall	Key Finding
AURORA (full)	**94.3%**	**156**	**Baseline**	**Complete system**
LLM reasoning	87.2%	142	**-7.1%**	LLM adds significant value
Neural Granger	91.5%	134	-2.8%	Temporal modeling crucial
GNN embeddings	89.8%	128	-4.5%	Graph learning helps
Adaptive partitioning	92.3%	178	-2.0%	Adaptation provides 2% boost
Multi-modal fusion	90.1%	149	-4.2%	Multiple sources critical
Bayesian uncertainty	93.1%	153	-1.2%	Uncertainty improves confidence
Only hierarchical RCD	89.0%	142	-5.3%	Matches published RCD
Only PC algorithm	86.0%	1247	-8.3%	Baseline without hierarchy
Only rule-based	73.1%	1.5	-21.2%	Traditional inadequate

E. Analysis and Taxonomy of Failure Modes

Table 5 shows how the performance goes down depending on what kind of failure it is. A single root cause is behind 97.2% of events, which is 60% of all events—baseline performance. Cascading failures are at 83.4%, which is 10.9% lower than 15%. Multi-root gets 78.1% (10%), which is 16.2% less than that. There are 71.5% (10%) Byzantine defects, which is a drop of 22.8%. Novel failures are at 67.3% (5%), which is 27.0% less

ISSN: 2178-7727

than before. The confusion matrix shows that the precision is 88.7%, the recall is 94.0%, and the F1-score is 91.3%. False Negative Analysis: 3 instances of cascading failure, 2 instances of unique failure, and 1 instance of incomplete observability. False Positive Analysis: Eight instances of strongly correlated non-causal services, three instances of erroneous correlation, and one instance of LLM hallucination.

Table 5: Performance Deterioration by Failure Category

Failure Type	Frequency	Top 5 Recall	95% CI	Performance vs Avg
Single root cause	60%	**97.2%**	[96.1%, 98.3%]	+2.9% (baseline)
Cascading failures	15%	**83.4%**	[80.2%, 86.6%]	-10.9% (harder)
Multi-root	10%	**78.1%**	[73.8%, 82.4%]	-16.2% (limitation)
Byzantine faults	10%	**71.5%**	[66.4%, 76.6%]	-22.8% (adversarial)
Novel failures (zero-shot)	5%	**67.3%**	[60.1%, 74.5%]	-27.0% (out-of-distribution)

F. Analysis of Confidence Calibration

Expected Calibration Error: AURORA 0.043 (well-calibrated), RUN 0.089 (overconfident) [10][21], RCD 0.156 (severely miscalibrated) [5][9]. AURORA's 90% confidence forecast is correct about 88% of the time. It indicates that Bayesian uncertainty quantification provides dependable confidence scores for decision-making involving human participation [25].

G. Variance Decomposition Analysis

ANOVA shows that the type of failure is to blame for 46.9% of the problems, the choice of approach is to blame for 24.9%, the size of the system is to blame for 18.7%, and the residual is to blame for 9.5%. The kind of failure is very important because new failures are always harder. It is very important to pick the right way to do things, and comparatively to baselines, AURORA has a big effect [5][10]. Moderate system size—it's important to be able to scale, but it's not the most important thing.

H. Analysis of Hyperparameter Sensitivity

Table 6 illustrates sensitivity to configuration parameters. RCD α sensitivity ±2.3% (poor) [5]. RCD subgroup size sensitivity is ±1.8% (low) [5][9]. LLM temperature sensitivity is ±8.4% (high) [11][27]. LLM maximum iterations sensitivity ±7.1% (high) [11] [12]. GNN hidden dimension sensitivity ±2.9% (minimal). Sensitivity of Neural Granger epochs is ±3.2% (medium) [10]. Proposed Configuration: RCD α = 0.05, subset = 20 [5][9]; LLM temperature = 0.0, max_iterations = 15 [27]; GNN hidden_dim = 64, Neural Granger epochs = 100 [10].

Table 6: Sensitivity Analysis of Configuration Parameters

Parameter	Range Tested	Optimal	Sensitivity (Δ Recall at ±50%)
RCD α (significance)	[0.01, 0.05, 0.10]	**0.05**	±2.3% (low) [5]
RCD subset size	[10, 20, 30, 50]	**20**	±1.8% (low) [5][9]
LLM temperature	[0.0, 0.3, 0.7, 1.0]	**0.0**	±8.4% (high) [11][27]
LLM max iterations	[5, 10, 15, 20]	**15**	±7.1% (high) [11][12]
GNN hidden dim	[32, 64, 128, 256]	**64**	±2.9% (low)
Neural Granger epochs	[50, 100, 200]	**100**	±3.2% (medium) [10]

ISSN: 2178-7727

PRODUCTION DEPLOYMENT CASE STUDY

A. Implementation

System Profile: 247 microservices for payments, authentication, and fraud detection; a mix of on-premises and AWS EKS Kubernetes; Prometheus (18,000 metrics), Elasticsearch (2TB of logs per day), and Jaeger (5 million traces per day); deployment in stages over six months.

Phase 1: Shadow Mode (Months 1–2) 156 incidents were examined, with 89.1% aligning with the assessments of a human physician. AURORA took an average of 2.7 minutes, while a person took an average of 38 minutes, which is a 93% drop.

Phase 2: Human-in-the-Loop (Months 3–4) 94 proposals were made; 85% of people were sure they would work; 87% of people accepted them; and 12 people turned them down because of business limits or conflicts.

Phase 3: Full Automation (Months 5–6) Low-risk activities happen on their own, medium-risk activities need permission, and high-risk activities always need permission. The average time it took to fix a problem (MTTR) went down by 91%, from 47 minutes to 4.3 minutes. The company saves about $1.2 million a year, and the false positive rate is 5.2%.

B. Cost-Benefit Evaluation

Infrastructure Expenses*:* The Kubernetes cluster costs $900 a month, Prometheus/Grafana costs $250 a month, the OpenAI API costs about $150 a month, and storage costs $80 a month. The total is about $1,380 a month (about $16,560 a year).

Advantages: The lower costs for engineers save $30,000 a year, the lower costs for downtime save about $1.2 million a year, and the total benefit is about $1.23 million a year.

ROI: ($1.23M - $16.5K) / $16.5K = 74-fold return on investment.

C. Scalability Assessment Under Load

Load testing results indicate: The load test showed that the average time for one user was 156 seconds, the 95th percentile time was 187 seconds, and the success rate was 100%. The average time for 10 users at once was 189 seconds, the 95th percentile time was 241 seconds, and the success rate was 98.7%. The average time for 20 users was 267 seconds, the 95th percentile time was 358 seconds, and the success rate was 95.2%. The average time for 50 users was 512 seconds, the 95th percentile time was 749 seconds, and the success rate was 87.3%.

Bottleneck Analysis*:* Incidents 1-10 show good scalability with hierarchical design [5][9][11]; incidents 10-20 have problems because of LLM API rate limits [11][27]; incidents 20-50 have problems because of Kubernetes API throttling [18], and causal inference is CPU-bound.

D. Operational Insights Acquired

Successful Aspects*:* Gradual deployment [25] built trust with the SRE team; explainability [5][11] made it easier for operators to understand decisions; human-in-the-loop [25] found edge situations and business limits; and integration with current tools [13-16] made sure that the adoption went smoothly.

Challenges: It was hard to integrate because OpenTelemetry collectors needed to be set up in a special way. Alert noise: The first false positive rate was 18%, but it had to be lowered to 5.2%. Change in the organization: Some operators didn't want automation; edge cases—unprecedented failure patterns—needed human diagnosis.

Optimal Strategies: Start with actions which are very safe and very confident [25] and then build trust over time [4]; Be sure to write down all your choices; Watch out for how often false positives and false negatives

ISSN: 2178-7727

happen; Create a way for people to give feedback that adds their corrections to the training set [28].

DISCUSSION

A. Under What Circumstances Does AURORA Achieve Success?

Criteria for Success: (1) Adequate observability [13][14][15][16] with less than 5% loss; (2) the causal Markov condition is satisfied [31] without significant unobserved confounders [5]; (3) Ample training data—at least 100 historical episodes [10][28]; (4) Moderate system scale—between 15 and 500 services [5][9]; (5) Singular root cause [5]. AURORA attains a 97.2% top 5 recall for single root cause failures on Sock-shop with full observability.

B. Under what circumstances does AURORA fail?

Novel Failures: 5% incidence; 67.3% recall compared to 94.3% overall; Supervised components lack generalizability [7][8]; Mitigation: Few-shot learning [24] with human input [25] and active learning.

Incomplete Observability*:* 8% frequency; 71.8% recall; The causal graph exhibits absent edges, contravening Theorem 2 [31]; Mitigation: Uncertainty quantification identifies low-confidence predictions. Warning: if observability gaps or confounding factors exist, identifiability assurances are invalid; the method relies on confidence criteria.

Cascading Failures: 15% incidence; 83.4% recall; the algorithm presumes a singular root cause [5][9]; *Mitigation:* Expand RCD to accommodate multiple root causes, temporal graph analysis [10].

LLM hallucinations occur at a frequency of 3%; incorrect remediation actions are taken; LLMs lack grounding [5][11]; and mitigation strategies include constraint-based verification, causal support [31] requirements, and human approval [25].

Byzantine Faults: 10% occurrence; 71.5% recall; Causal discovery presupposes truthful data [5][31]; Mitigation strategies include robust statistics, outlier detection, and multi-modal cross-validation [33].

C. Constraints and Prospective Research

The hierarchical approach to computational complexity [5][9] attains $O(n^2 \log n)$ [31] compared to $O(n^3)$ for PC [5], although processing over 1000 services necessitates approximately 15 minutes [Table 3]. Future: heuristic algorithms. (theoretical bound O(n^2 log n); practically I see near-quadratic scaling)

Causal Graph Accuracy: May contain erroneous edges [7][8][20], propagating errors. 16/29 algorithm setups caused errors [20]. Mitigation: ensemble techniques integrating PC [5], RCD [5][9], and neural Granger [10][21][33].

Unprecedented Failure Modes: Difficulty with novel failures [7][8] (67.3% compared to 94.3% [Table 4]). Future: meta-learning for few-shot adaptation.

Multi-Root Causality: Current Theorem 2 [31] presupposes a singular root cause [5]. Actual occurrences (10% [Table 4]) entail numerous failures. Future: expand to the k-root scenario [31][32]. Warning: Identifiability promises are invalid in the presence of observability gaps or confusion; the system relies on confidence thresholds.

D. Practical Implications for SRE and Platform Engineering

The results show that AURORA works best when certain conditions are met: (1) high observability, which means stable OTel/Prometheus/Jaeger pipelines and deploy-event capture [13–16]; (2) actions that are confidence-gated and need human approval [25]; and (3) the agent being the main user of CI/CD, with signals and recommendations for autoscaling that can be changed as needed [18]. Your research showed that the top five recollections were the strongest when Sock-Shop was fully displayed. This backs up the assumption that

ISSN: 2178-7727

the best way to get the most out of telemetry is to get good telemetry. The LLM ReAct supervision sped up triage a lot with ensemble causal discovery (PC/RCD/Neural-Granger) while still being easy to grasp [5] [9–11].

E. Threats to Validity (internal, external, construct)

Internal: When there are partial observability or hidden confounders, the assurances of identifiability become weaker. In these situations, the technique must rely on calibrated confidence and human evaluation instead of proofs [31][5][25]. Even with the correct localization, LLMs can still get remediation wrong; checks and approvals for constraints are needed [11][27].

External: Gains like 94.3% top-5 recall may change depending on prior failures and runbook styles. However, the hierarchical/ensemble architecture helps to reduce domain shift. [5] [9-10]. Top-k measures don't entirely show how much trust an operator has in a firm or how risky it is; production HITL approval rates and false-positive monitoring help fill in some of the gaps [25].

F. Generalizability and Scalability.

Your hierarchical method has a complexity of $O(n^2 \log n)$, which is better than the classical PC's $O(n^3)$ complexity. This means that mid-sized to huge estates are manageable [31][5][9]. With more than 1000 services, analysis time (~15 min) proposes partitioned graphs and a coarse-to-fine search for real-time application. Ensemble discovery also makes things less fragile, which is significant because there is evidence that 16 out of 29 algorithm configurations can add wrong edges [20].

G. Human Factors and Governance

Operational risk was mostly about faith in automation, policy exceptions, and dealing with edge cases. The staggered rollout—shadow → human-in-the-loop → guarded automation—along with explanation artifacts (causal subgraphs + NL rationales) worked well for managing change [25][11]. I suggest policy-as-code guardrails that connect action classes to confidence bands, keep audit trails that can't be changed, and always learn from operator overrides [25, 13-14].

H. Future Research Directions

1. **Few/zero-shot novel failures:** Accuracy reduces with novel failures; investigate meta-/few-shot adaptations and retrieval that integrate change events with telemetry deltas. [24] [25].
2. **Causality under partial observability:** Add priors from deployment manifests and SLOs and make the confidence calibration tighter to be safe when edges are missing [31][33].
3. **Multi-root and cascading failures:** Expand hierarchical search to list and rank multi-cause temporal hypotheses, considering production cascades [10].
4. **End-to-end evaluation:** Set standard standards that combine localization, action selection, and business results (MTTR/rollback accuracy) with top-k metrics [5][10][25].

FINAL ASSESSMENT

AURORA is a production-grade multi-agent framework that integrates LLM-powered ReAct reasoning with hierarchical causal discovery and neural Granger causality, attaining a top 5 recall of 94.3%, surpassing RUN (91%) and RCD (89%) with statistical significance ($p < 0.001$, Cohen's $d = 0.85$).

Key Contributions: (1) Theoretical Framework—Innovative formal foundations [31][32][33] for causal discovery in non-stationary environments with $O(n^2 \log(n))$ sample complexity (Theorem 1), identifiability conditions (Theorem 2), intervention detection (Lemma 1), and multi-modal fusion guarantees (Theorem 3); (2) Multi-Agent Architecture—Integration of the ReAct pattern [11] [12][27] with seven specialized agents [26], adaptive hierarchical RCD incorporating Bayesian uncertainty; (3) Extensive Empirical Validation—Over 500 failure scenarios across 15-1000 services, ablation studies (Table 4), failure taxonomy (Table 5), ECE = 0.043, and 46.9% variance in failure types; (4) Production Deployment—Implementation of 247 microservices, achieving a 91% reduction in MTTR (from 47 minutes to 4.3 minutes), with 67 automated

resolutions and a 74× ROI; (5) Failure Analysis—Quantified performance metrics: novel failures (67.3%), cascading failures (83.4%), and Byzantine failures (71.5%). Warning: if there are observability gaps or confounding factors, identifiability assurances are invalid; the method relies on confidence thresholds.

Practical Impact: It uses causal inference to find the cause of the problem, which cuts the time it takes to respond to incidents from 3 hours (manual [5]) to less than 5 minutes [5][31]. Auto-scaling integration helps meet service-level goals and makes better use of resources.

Theoretical Contribution: Initial formal examination of hierarchical causal discovery complexity (Theorem 1) [31], criteria for identifiability (Theorem 2) [31][32], and assurances for multi-modal fusion (Theorem 3) [33]. Warning: if observability gaps or confounding factors exist, identifiability assurances are invalid; the method relies on confidence criteria.

Empirical Rigor: Systems encompassing 15-1000 services and surpassing 500 failure types, whereas in previous studies that examined a maximum of 41 services and fewer than 100 instances. Bootstrap confidence intervals, effect sizes, ablation studies, and variance decomposition delineate performance thresholds.

Future Directions: Proactive failure prediction; multi-root causality theory and methods; Transfer learning for few-shot adaptation; Formal verification for remediation safety; Federated learning with differential privacy. Full implementation is accessible at https://github.com/jay-gatech/agenticai-rca-langchain.git [30].

ACKNOWLEDGMENTS

I express our gratitude to the authors of the RCD algorithm [5][9] (Ikram et al., NeurIPS 2022), the authors of the RUN technique [10][21] (Lin et al., AAAI 2024), and the LangChain community [27]. Gratitude is extended to Grafana Labs for their help on OpenTelemetry and Prometheus, Microsoft for providing implementation examples, and the Kubernetes community. Appreciative of the SRE team's involvement and feedback during the production deployment.

References

[1] IEEE Xplore, “Microservices: architecture, containers, and challenges,” in *Proc. IEEE Conf.*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9282637/

[2] IEEE Xplore, “A Comprehensive Study of Microservices Architecture in Cloud Computing,” in *Proc. IEEE Conf.*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10959671/

[3] IEEE Xplore, “A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges,” *IEEE Access*, vol. 11, pp. 1–25, 2023. [Online]. Available: https://ieeexplore.ieee.org/iel7/6287639/10005208/10220070.pdf

[4] IEEE Chicago, “Microservices Design Patterns for Cloud Architecture,” *IEEE Chicago Newsletter*, Sept. 2024. [Online]. Available: https://ieeechicago.org/microservices-design-patterns-for-cloud-architecture/

[5] A. Ikram, S. Chakraborty, S. Mitra, S. K. Saini, S. Bagchi, and M. Kocaoglu, “Root Cause Analysis of Failures in Microservices through Causal Discovery,” in *Proc. 36th Conf. Neural Information Processing Systems (NeurIPS)*, 2022, pp. 1–13. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf

[6] IEEE Xplore, “Challenges in Documenting Microservice-Based IT Landscape,” in *Proc. IEEE Int. Conf. Software Architecture*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8944979/

[7] L. Pham, H. Ha, and H. Zhang, “Root Cause Analysis for Microservices Based on Causal Inference: How Far Are We?” arXiv *preprint arXiv:2408.13729*, 2024. [Online]. Available: https://arxiv.org/pdf/2408.13729.pdf

[8] L. Pham, H. Ha, and H. Zhang, “Root Cause Analysis for Microservices Based on Causal Inference,”

arXiv preprint arXiv:2408.13729, ver. 2, 2024. [Online]. Available: https://arxiv.org/html/2408.13729v2

[9] NeurIPS, "Supplemental Material: Root Cause Analysis of Failures in Microservices through Causal Discovery," in *Proc. NeurIPS*, 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Supplemental-Conference.pdf

[10] C.-M. Lin, C. Chang, W.-Y. Wang, K.-D. Wang, and W.-C. Peng, "Root Cause Analysis In Microservice Using Neural Granger Causal Discovery," in *Proc. 38th AAAI Conf. Artificial Intelligence*, 2024. [Online]. Available: https://arxiv.org/abs/2402.01140

[11] LangGraph Team, "Agent architectures—LangGraph Concepts," *GitHub Pages*, 2025. [Online]. Available: https://langchain-ai.github.io/langgraph/concepts/agentic_concepts/

[12] LangGraph Team, "Agent architectures—L—LangGraphConceptsS," *GitHub Pages*, 2025. [Online]. Available: https://langchain-ai.github.io/langgraphjs/concepts/agentic_concepts/

[13] Grafana Labs, "Introducing an OpenTelemetry Collector distribution with built-in Prometheus pipelines: Grafana Alloy," *Grafana Blog*, Apr. 2024. [Online]. Available: https://grafana.com/blog/2024/04/09/grafana-alloy-opentelemetry-collector-with-prometheus-pipelines/

[14] Microsoft, "Example: Use OpenTelemetry with Prometheus, Grafana, and Jaeger," *Microsoft Learn*, Aug. 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-prgrja-example

[15] Prometheus Authors, "Overview - Prometheus," *Prometheus Documentation*, 2024. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[16] Grafana Labs, "Using OpenTelemetry and Prometheus: A practical guide to data collection," *Grafana Blog*, Nov. 2024. [Online]. Available: https://grafana.com/blog/2023/07/20/a-practical-guide-to-data-collection-with-opentelemetry-and-prometheus/

[17] Rafay Systems, "Auto-Scaling K8s Applications on Amazon EKS: 8 Best Practices," *Rafay Blog*, Feb. 2024. [Online]. Available: https://rafay.co/ai-and-cloud-native-blog/auto-scaling-kubernetes-applications-amazon-eks

[18] Kubernetes Authors, "Autoscaling Workloads - Kubernetes," *Kubernetes Documentation*, Apr. 2025. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/autoscaling/

[19] GlobalLogic, "Optimizing Your Microservices Architecture With Auto-Scaling," *GlobalLogic White Paper*, 2021. [Online]. Available: https://www.globallogic.com/wp-content/uploads/2021/03/Optimizing-your-micro-services-architecture-with-auto-scaling.pdf

[20] D. Markakis et al., "From Logs to Causal Inference: Diagnosing Large Systems," *Proc. VLDB Endowment*, vol. 18, pp. 158–171, 2025. [Online]. Available: https://www.vldb.org/pvldb/vol18/p158-markakis.pdf

[21] zmlin1998, "RUN: Official repository of Root Cause Analysis In Microservice Using Neural Granger Causal Discovery," *GitHub*, 2023. [Online]. Available: https://github.com/zmlin1998/RUN

[22] Quotient AI, "Evaluating Tool Calling Capabilities in Large Language Models: A Literature Review," *Quotient AI Blog*, May 2025. [Online]. Available: https://blog.quotientai.co/evaluating-tool-calling-capabilities-in-large-language-models-a-literature-review/

[23] IBM Research, "Large language models revolutionized AI. LLM agents are what's next." *IBM Research Blog*, Feb. 2021. [Online]. Available: https://research.ibm.com/blog/what-are-ai-agents-llm

[24] NVIDIA Research, "Small Language Models are the Future of Agentic AI," *NVIDIA LPR*, Dec. 2024. [Online]. Available: https://research.nvidia.com/labs/lpr/slm-agents/

[25] Anthropic, "Building Effective AI Agents," *Anthropic Research*, Dec. 2024. [Online]. Available: https://www.anthropic.com/research/building-effective-agents

[26] Y. Zhang, "Agent-as-Tool: A Study on the Hierarchical Decision Making with Reinforcement Learning," *arXiv preprint arXiv:2507.01489*, 2025. [Online]. Available: https://arxiv.org/html/2507.01489v1

[27] LangChain Team, "Build an Agent," *LangChain Tutorials*, June 2025. [Online]. Available:

ISSN: 2178-7727

https://python.langchain.com/docs/tutorials/agents/

[28] H. Qiu et al., "Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems," in *Proc. USENIX ATC*, 2023, pp. 1–15. [Online]. Available: https://www.usenix.org/system/files/atc23-qiu-haoran.pdf

[29] R. Chen et al., "Streamlining Resilient Kubernetes Autoscaling with Multi-Agent Systems," *arXiv preprint arXiv:2505.21559*, 2025. [Online]. Available: https://arxiv.org/pdf/2505.21559.pdf

[30] AgenticAI-RCA-Langchain [Source code]. Available: https://github.com/jay-gatech/agenticai-rca-langchain.git

[31] J. Pearl, "Causality: Models, Reasoning, and Inference," Cambridge University Press, 2009.

[32] P. Spirtes, C. Glymour, and R. Scheines, "Causation, Prediction, and Search," MIT Press, 2000.

[33] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Toward Causal Representation Learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 612–634, 2021.