Cloud Microservices: An Empirical Study of MCP-Orchestrated LLM Agents for Incident Triage

Reinan Gabriel Dos Santos Souza[1], **Methanias Colaço Júnior**[1]

[1] Graduate Program in Computer Science (PROCC)
Federal University of Sergipe (UFS)
Av. Marechal Rondon, s/n – Jardim Rosa Elze – Postal Code 49100-000
São Cristóvão – SE – Brazil

reinangabriel1520@gmail.com, mjrse@hotmail.com

***Abstract. Context**: The growing complexity of cloud microservices imposes significant challenges for Site Reliability Engineering (SRE), contributing to delayed incident resolution and increased operational effort. **Objective**: This study evaluated the effectiveness of autonomous agents based on Large Language Models (LLMs), orchestrated via the Model Context Protocol (MCP), for root cause analysis in a cloud-native setting. **Method**: We conducted a controlled Randomized Complete Block Design (RCBD) experiment in Kubernetes with automated fault injection, covering three distinct failure scenarios and multiple LLM configurations across 360 executions. **Results**: A high-performing configuration (Gemini 2.5 Flash at low temperature) achieved a* $71.1\,\%$ *root-cause identification success rate, substantially above a random-chance baseline (*$\approx 0.91\%$*). Smaller models exhibited higher token and step volatility (*$CV = 2.17$*) and more repeated tool-call cycles, challenging the assumption that lower-parameter models are inherently more cost-effective for SRE workflows. **Conclusion**: The results provide empirical evidence that MCP-orchestrated LLM agents can support root cause analysis in cloud-native environments and offer practical guidance for model selection in AIOps/SRE workflows.*

1. Introduction

The increasing adoption of microservices architectures in cloud environments introduces complexity for Site Reliability Engineering (SRE), as service interconnection allows failures to propagate throughout the system, hindering prompt Root Cause Analysis (RCA) [Bagehorn et al. 2022]. An industry report published by New Relic[1] highlights the severity of financial consequences, estimating that companies face an average annual cost of US$ 76 million due to high-impact Information Technology (IT) outages. Although this figure originates from a non-peer-reviewed source, it illustrates the economic magnitude that motivates the development of automated diagnostic approaches. In this context, the resilience of cloud-native systems is a pillar of digital transformation, as failures in critical infrastructures directly impact the capacity to respond to climate emergencies and other crisis scenarios.

To address these challenges, Artificial Intelligence for IT Operations (AIOps) has evolved toward autonomous agent-based approaches, termed Agentic AIOps

[1] https://newrelic.com/press-release/20250917

[Zota et al. 2025], where the concept of Cognitive Observability extends traditional monitoring to capture reasoning traces and decision-making signals from autonomous agents [Rombaut et al. 2025]. In this paradigm, the Model Context Protocol (MCP) provided the standardized framework for LLM-tool orchestration in this study, reducing ambiguities and excessive token consumption [Yang et al. 2025, Xu et al. 2025a]. However, applying LLMs to RCA tasks presents risks due to hallucinations, imprecise parameter extraction, and tool invocation failures [Pei et al. 2025].

Previous studies, such as [Xu et al. 2025b], have established benchmarks for evaluating Large Language Model (LLM) reasoning in failure scenarios. Additionally, [Pham et al. 2025] introduces datasets and an evaluation environment forRCA in microservice systems. Despite these advancements, there is a notable gap in the literature concerning the influence of agent configuration parameters on diagnostic stability and economic efficiency. Furthermore, experimental evaluations that quantify the trade-off between exploration and exploitation in controlled environments are limited.

This study evaluated LLM-based autonomous agents in RCA across three dimensions: (i) diagnostic efficacy, (ii) economic efficiency, and (iii) operational stability. A controlled experiment using a Randomized Complete Block Design (RCBD) compared Google Gemini variants (generations 2.0 and 2.5) in a Kubernetes environment with the Online Boutique[2] application, orchestrated via MCP. Results indicated that *Gemini 2.5-Flash* at low temperature achieved a $71.1\,\%$ success rate ($p < 0.001$), while lower-parameter models exhibited a *hidden cost of incompetence*, with operational instability ($CV = 2.17$) and higher total costs due to redundant reasoning cycles.

The remainder of this paper is organized as follows. Section 2 presents the theoretical background. Section 3 discusses the related work. Section 4 describes the research design, variables, and experimental procedures. Section 5 details the experimental environment and fault-injection scenarios. Section 6 reports and discusses the results. Section 7 outlines the threats to validity, and Section 8 concludes the paper with final remarks and future work.

2. Theoretical Background

2.1. Agentic AIOps

Enabling autonomous cloud infrastructures depends on implementing AIOps agents capable of identifying, tracking, and remediating faults automatically, reducing the need for direct human intervention [Shetty et al. 2024]. Unlike conventional AIOps systems, which rely on predefined rules and reactive responses, Agentic AIOps represent an evolutionary step by integrating autonomous Artificial intelligence (AI) agents that proactively monitor, analyze, and respond to IT events in real time [Zota et al. 2025].

In the present study, this paradigm is operationalized through an LLM-based cognitive agent that autonomously investigates an incident in a Kubernetes environment. The agent follows a ReAct reasoning loop [Yao et al. 2023], interacting with observability tooling via MCP to perform RCA without direct human intervention. This design instantiates the Agentic AIOps vision within a controlled experimental setting, enabling the quantification of diagnostic efficacy, economic efficiency, and operational stability.

[2]https://github.com/GoogleCloudPlatform/microservices-demo

2.2. Model Context Protocol

MCP has been proposed as a standardized protocol for LLM-tool interaction. According to [Li et al. 2025], it defines structured data formats and communication semantics, and improves cross-platform compatibility and robustness by abstracting tool descriptions, input/output schemas, and permission metadata. The protocol also enables LLMs to autonomously discover and invoke tools in scenarios involving diverse and dynamic tasks.

In this study, MCP provides the technical foundation to interact with Online Boutique failure scenarios in a model-agnostic manner. This standardization mitigates tool invocation ambiguities and ensures diagnostic interface reproducibility across different model variants.

3. Related Work

OpenRCA [Xu et al. 2025b] introduced a public benchmark for LLM-based RCA over 335 real-world failures with more than 68 GB of telemetry data, proposing an execution-based multi-agent approach. Even its best configuration achieved only 11.34% accuracy, underscoring the difficulty of the task. While OpenRCA advances benchmark-oriented feasibility evaluation, our study conducts a controlled cloud-native experiment that quantifies diagnostic success, tool-use cost, and execution stability under systematic parametric variation.

L4 [Jiang et al. 2025] addresses failure diagnosis in LLM training platforms through automated log analysis, achieving an 87.3% F1-score for failure-indicating log identification and 80% top-5 accuracy for faulty-node localization. The scope of this approach is limited to training infrastructure and log-centric methods. In contrast, the present study focuses on cloud-native microservice triage, where agents interact with observability tools via the MCP, emphasizing the trade-offs among diagnostic performance, token consumption, and operational stability.

MicroRemed [Zhang et al. 2025] investigates end-to-end microservice remediation, where LLMs generate executable Ansible playbooks from diagnosis reports across 421 fault-recovery pairs. In contrast, our study focuses on the diagnostic process itself, without presuming pre-existing diagnoses. We uniquely measure how accurately and efficiently MCP-orchestrated agents identify root causes under repeated controlled executions.

Flow-of-Action [Pei et al. 2025] is the closest work to ours in diagnostic intent, proposing an SOP-enhanced multi-agent architecture that mitigates hallucinations and reports substantial gains over ReAct-style baselines. Rather than proposing a new agent architecture, our contribution is empirical: we examine how MCP-based orchestration and model-level parameters affect diagnostic success, cost, and stability in incident triage.

[Xu et al. 2025a] improve MCP-based Kubernetes orchestration through schema-constrained decoding, achieving major reductions in latency and token usage while preserving command validity. This work strengthens the orchestration substrate but does not evaluate diagnostic reasoning. Our study builds on the same MCP ecosystem while moving the focus from command-generation efficiency to end-to-end diagnostic evaluation.

CausalRCA [Xin et al. 2023] formulates root cause localization as a causal inference problem, combining gradient-based causal structure learning with graph-based

inference and reporting an average Avg@5 improvement of 9.43%. Compared with this causal-graph perspective, our work adopts an agentic approach: LLM agents reason over raw observability signals via MCP without requiring pre-constructed causal models, with explicit measurement of operational cost and variability.

Table 1 provides a comparative summary of the related work, ordered by operational stage from upstream localization to downstream remediation.

Table 1. Comparative Summary of Related Work

Work	Focus	Method	Key Metrics*
[Xin et al. 2023]	Causal root-cause	Gradient-based causal inference	AC@k, Avg@k
[Jiang et al. 2025]	Failure diagnosis	Automated log analysis	F1, Recall
[Xu et al. 2025b]	RCA benchmark	Multi-agent execution	Accuracy
[Pei et al. 2025]	Microservice RCA	SOP-guided multi-agent	LA, TA, APL
[Xu et al. 2025a]	MCP orchestration	Schema-constrained decoding	Latency, tokens
[Zhang et al. 2025]	Remediation	Multi-agent + playbooks	Accuracy, latency
This study	**Incident triage**	**RCBD experiment**	**SR, CV, FPR**

**AC@k = Accuracy at rank k; Avg@k = Average score at rank k; LA = Root Cause Location Accuracy; TA = Root Cause Type Accuracy; APL = Average Path Length; SR = Success Rate; CV = Coefficient of Variation; FPR = False Positive Rate.*

As summarized in Table 1, the related work is ordered from upstream causal localization to downstream remediation, reflecting the stages of the incident management lifecycle. Two observations emerge from this comparison. First, existing studies that address diagnostic accuracy [Xu et al. 2025b, Pei et al. 2025, Xin et al. 2023] do not examine how model-level configuration parameters, such as capacity and inference temperature, affect operational cost and execution stability. Second, the only prior work operating within the MCP ecosystem [Xu et al. 2025a] focuses on command-generation efficiency rather than end-to-end diagnostic reasoning.

4. Methodology

The methodology followed an explanatory research design [Severino 2018], operationalized through a controlled experiment. In accordance with the experimental development process for AI applications [Colaço Júnior et al. 2022, Colaço Júnior 2025], the approach moved beyond exploratory analysis by manipulating independent variables, specifically fault injection and agent parametric configurations, to isolate and quantify causal relationships affecting the efficacy and efficiency dependent variables.

The experiment followed four phases: (1) *Prepare Experiment*, in which infrastructure was provisioned with Terraform on GKE to ensure reproducibility; (2) *Execute Experiment*, in which faults were injected via Istio *VirtualService* resources or Deployment environment variables, depending on the scenario; (3) *Gather Data*, comprising the collection of reasoning traces and tool calls; and (4) *Define and Perform Data Validation*, in which RCA outcomes were validated against a deterministic Ground Truth annotated by an SRE engineer.

4.1. Experiment Design

A RCBD was adopted to control for variability across fault scenarios, which served as the blocking variable. This design isolates the effects of agent configuration parameters on

diagnostic outcomes while accounting for scenario-level heterogeneity. Each configuration was evaluated across 15 independent repetitions per block, totaling 360 experimental runs (4 models $\times$ 2 temperatures $\times$ 3 scenarios $\times$ 15 repetitions).

4.2. Experiment Goal

The goal of this experiment is to evaluate LLM-based autonomous agents orchestrated via MCP for RCA in cloud-native microservice environments, with respect to diagnostic efficacy, economic efficiency, and operational stability, from the viewpoint of SRE teams, in the context of fault-injection scenarios on the Online Boutique application deployed on Google Kubernetes Engine (GKE).

Three Research Questions (RQs) guided the evaluation. **RQ1** examines diagnostic efficacy through the Success Rate (SR), a binary metric requiring correct identification of both fault location and type, validated by an SRE engineer against a deterministic Ground Truth. **RQ2** investigates the trade-off between inference configuration and resource consumption, measured by total token count (C_{tokens}) and total tool calls (N_{steps}). **RQ3** assesses operational stability via the Coefficient of Variation (CV) across iterations.

4.3. Dependent and Independent Variables

The independent variables were: (i) Agent Configuration, manipulated via LLM temperature at two levels, Conservative ($T_{low} = 0.1$) and Creative ($T_{high} = 0.9$); and (ii) Fault Scenario (F_n), serving as the blocking variable across three operational contexts (F_1 to F_3). The experiment tested whether agent efficacy significantly exceeded a random-chance baseline of $\approx 0.91\,\%$ (computed as $1/11$ services $\times$ $1/10$ fault categories), and whether temperature affected cost and stability.

4.4. Selection of Objects

Four variants from the Google Gemini Flash model family were selected, with selection limited to the Vertex AI catalog to ensure reproducibility. Table 2 presents the configurations, which span two generations (2.0 and 2.5) and two processing tiers (Lite and Standard).

Table 2. Selected Language Models for Experimental Evaluation

Model Family	Version	Tier
Gemini Flash-Lite	2.0	Lite
Gemini Flash	2.0	Standard
Gemini Flash-Lite	2.5	Lite
Gemini Flash	2.5	Standard

4.5. Instrumentation and Fault Scenarios

Three fault scenarios were designed to cover the full spectrum of fundamental microservice incidents. Table 3 summarizes each scenario, including the injection mechanism and Ground Truth criteria.

Scenario F_1 tests the agent's ability to correlate user-facing HTTP errors with network-layer fault injection configurations. F_2 requires the agent to distinguish

Table 3. Fault Scenarios for Experimental Evaluation

Scenario	Target Service	Injection Mechanism	Ground Truth
F_1	`shippingservice`	Istio VirtualService abort (HTTP 500, 90% of requests)	Identify service, VirtualService config, and injection rule
F_2	`productcatalog`	`EXTRA_LATENCY=4s` environment variable in Deployment	Identify env variable and exact delay value
F_3	None	No injection (false-premise user report simulating organizational noise)	Report system as healthy (no service attribution, hallucination = failure)

application-layer latency from infrastructure bottlenecks by inspecting Kubernetes Deployment manifests. F_3 serves as a negative control, evaluating robustness against hallucinations: the agent must refute a false incident report by confirming that observability signals (error rates, latency histograms, and log queries) indicate nominal system health.

5. Experiment Operation

5.1. Prepare Experiment

The experimental environment was provisioned using Terraform on Google Kubernetes Engine (GKE Standard, v1.33.5), ensuring infrastructure idempotency across iterations. The infrastructure source code is available in a public repository[3]. Figure 1 illustrates the logical architecture.

The target system is Online Boutique (v0.10.4), a polyglot microservices application comprising 11 services implemented in Go, Java, C#, Python, and Node.js, interconnected via synchronous gRPC dependencies. This architecture is established as a reference platform in microservice research [Zheng et al. 2024, Xiao et al. 2025, Zhang et al. 2025].

5.2. Execute Experiment

The observability stack captures the four Golden Signals (latency, traffic, errors, saturation) through Prometheus (metrics), Grafana (visualization), Grafana Loki (logs), and Grafana Tempo (distributed traces). Istio (v1.24.1) serves dual roles as service mesh and chaos engineering engine, injecting deterministic faults via VirtualService resources without application modifications. Execution was automated through a GitLab Runner

[3]https://github.com/ReinanHS/mcp-llm-incident-triage-infra

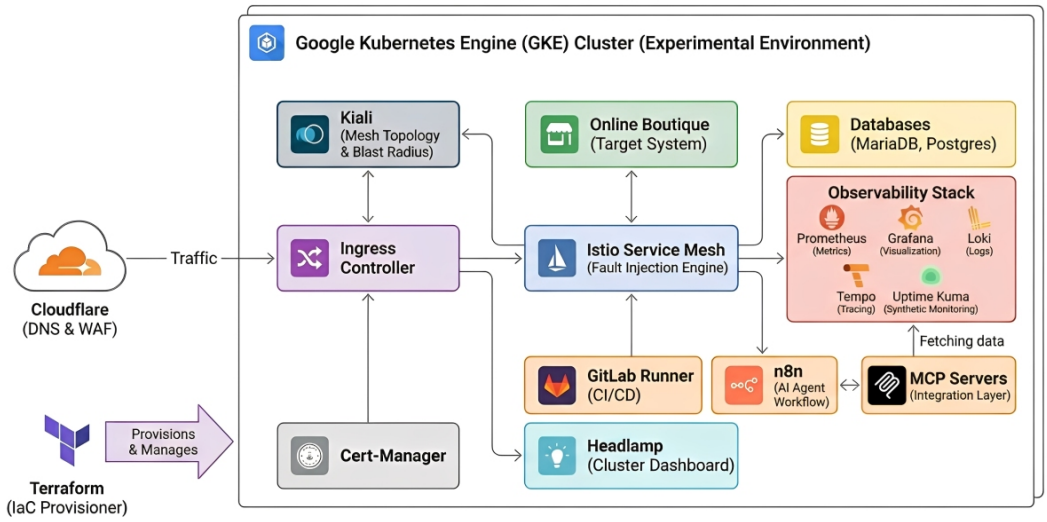

Figure 1. Experimental environment architecture on GKE, showing the target system, observability stack, service mesh, and MCP integration.

pipeline that orchestrates scenario provisioning, parallel model evaluation, and system teardown for each of 15 iterations per block.

The cognitive agent was developed on the n8n orchestration platform with MCP integration, connecting to *grafana-mcp* for metrics and logs, and to *mcp-server-k8s* for Kubernetes introspection. The agent follows a cyclic ReAct loop [Yao et al. 2023] across four stages: (1) initialization with scenario context and RFC3339 timestamps; (2) iterative investigation enforcing a hierarchical Standard Operating Procedure (SOP) (Global Metrics $\rightarrow$ Logs $\rightarrow$ Infrastructure); (3) cycle control with an $N = 10$ iteration limit; and (4) synthesis of a structured diagnostic report. A pilot study (228 preliminary runs) preceded official data collection, leading to prompt calibration, Application Programming Interface (API) failure resilience mechanisms, and rate-limiting strategies.

The dataset comprises $N = 360$ runs (4 models $\times$ 2 temperatures $\times$ 3 scenarios $\times$ 15 repetitions). Raw interaction data, including reasoning traces, tool calls, and token consumption, were captured and persisted in an analytical database and are publicly available in a companion repository[4].

6. Results and Discussion

6.1. Diagnostic Efficacy

Table 4 summarizes the results across 360 runs ($N = 45$ per group, 15 repetitions $\times$ 3 scenarios). *Gemini 2.5-Flash* at $T = 0.1$ achieved the highest Success Rate (71.1 %), while *2.0-Flash* at $T = 0.1$ recorded the lowest (2.2 %). A one-tailed binomial test confirmed that the best-performing agent significantly exceeds the random-chance baseline of $\approx 0.91\,\%$ ($p < 0.001$), validating that diagnostic performance stems from intrinsic inferential capability rather than randomness.

[4]https://gitlab.com/labchatops/data

Table 4. Efficacy and Economic Efficiency Ranking (RCBD Analysis)

Model	Temp.	SR (%)	Avg Tokens	Avg Steps	CV
2.5-Flash	0.1	71.1%	463,886	13.9	0.60
2.5-Flash	0.9	62.2%	375,582	12.4	0.52
2.0-Flash-Lite	0.1	35.6%	629,967*	16.3	0.62
2.0-Flash-Lite	0.9	28.9%	362,659*	10.1	0.57
2.5-Flash-Lite	0.1	22.2%*	287,682*	65.3	2.17
2.0-Flash	0.9	22.2%*	136,292	2.3	1.24
2.0-Flash	0.1	2.2%*	140,430	31.4	1.36
2.5-Flash-Lite	0.9	2.2%*	445,989*	4.9	0.81

Note: Table sorted by Success Rate (SR) in descending order. (*) indicates significant difference ($p < 0.05$) between temperatures of the same model, controlling for block effects via Mixed-Effects Models.

Figure 2 illustrates diagnostic efficacy (SR) across temperature levels using bar charts with $95\,\%$ confidence intervals (CIs). The high-capacity model (*Gemini 2.5-Flash*) achieved peak performance under determinism ($T = 0.1$). Conversely, *Gemini 2.0-Flash* exhibited divergent behavior, increasing success rates from $2.2\,\%$ to $22.2\,\%$ when transitioning to higher stochasticity ($T = 0.9$). *Lite* variants showed efficacy degradation as temperature increased, highlighting the instability of small-parameter architectures in exploratory reasoning processes.

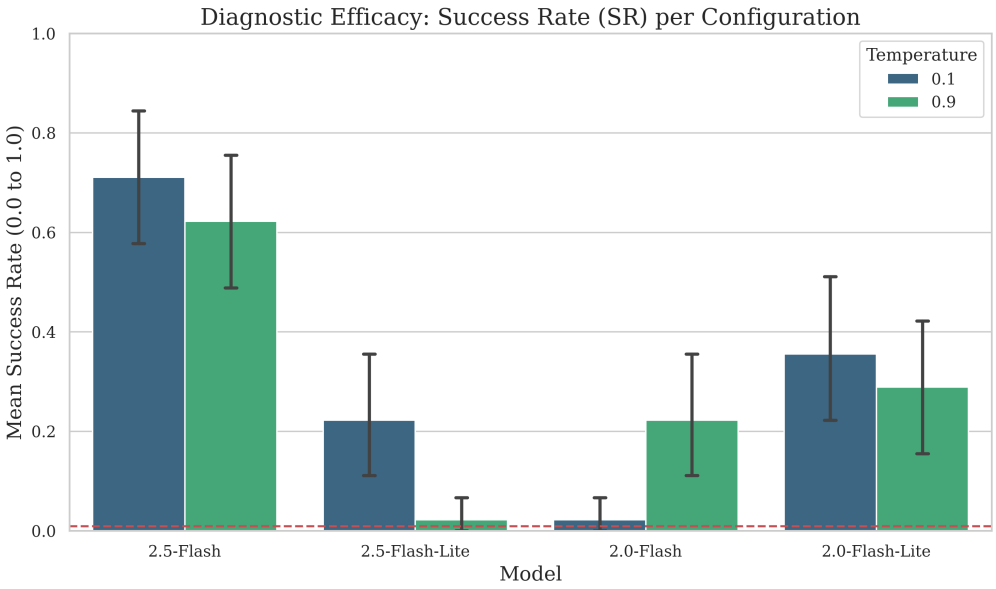

Figure 2. Diagnostic efficacy by model and temperature with 95% CIs. Determinism favors *Gemini 2.5-Flash*; smaller models exhibit high sensitivity to thermal variation.

6.2. Economic Efficiency and Operational Stability

Inspection of execution artifacts indicated that small-parameter models frequently entered redundant reasoning cycles when unable to identify root causes. This observation intro-

duces the concept of the hidden cost of incompetence: theoretical per-token savings from smaller models are negated by increased call volume resulting from limited analytical precision. For example, 2.0-Flash-Lite at $T = 0.1$ consumed an average of 629,967 tokens, the highest among all configurations, despite offering the lowest per-token cost. In contrast, Gemini 2.5-Flash demonstrated consistent token consumption and achieved a lower total cost per incident.

Operational stability, measured by the CV of execution steps, challenges the assumption that determinism guarantees consistency. *2.5-Flash-Lite* at $T = 0.1$ showed high volatility ($CV = 2.17$), due to diagnostic convergence failure in complex scenarios. Under determinism, models with lower reasoning capacity tend to repeat redundant investigation steps. In contrast, *Gemini 2.5-Flash* remained stable at both temperatures ($CV \leq 0.60$, mean of 13.9 steps), suggesting that stability is primarily due to model architecture robustness rather than temperature settings.

6.3. Trade-off Analysis and Negative Control

Figure 3 correlates mean execution cost with SR, identifying the efficiency frontier for decision-making in production environments. Analysis shows that *Gemini 2.5-Flash* ($T = 0.1$) establishes the efficacy upper bound, offering the highest Return on Investment (ROI) for root cause analysis tasks. While *Gemini 2.0-Flash* presents the lowest nominal cost, its low success rate excludes it from the efficiency frontier for critical operations.

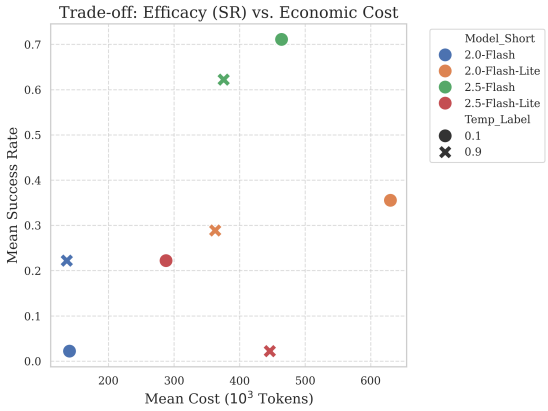

Figure 3. Cost vs. Efficacy Trade-off. *Gemini 2.5-Flash* dominates the high-efficacy region; *Lite* variants exhibit inefficiency from retry loops.

The negative control scenario (F_3, healthy system) assessed robustness to hallucinations. Table 5 presents configuration rankings by False Positive Rate (FPR). *2.0-Flash-Lite* ($T = 0.1$) achieved $0.0\,\%$ FPR across 15 runs, whereas *2.5-Flash* ($T = 0.1$) produced a moderate $13.3\,\%$. Notably, *2.0-Flash* recorded $93.3\,\%$ FPR regardless of temperature, and *2.5-Flash-Lite* at $T = 0.9$ reached error saturation ($100.0\,\%$ FPR), interpreting nominal metric fluctuations as systemic failure. These findings indicate that models with high diagnostic efficacy do not necessarily minimize hallucination rates, highlighting a substantive tension between sensitivity and specificity.

7. Threats to Validity

Internal validity: Although deterministic fault injection and automated state restoration mitigated confounding across iterations, the use of preemptible Spot VMs on GKE may

Table 5. Robustness Ranking: False Positive Rate (FPR)

Model	Temp.	N	FP	FPR (%)	95% CI
2.0-Flash-Lite	0.1	15	0	0.0%	[0.0, 20.4]
2.5-Flash	0.1	15	2	13.3%	[3.7, 37.9]
2.0-Flash-Lite	0.9	15	3	20.0%	[7.0, 45.2]
2.5-Flash	0.9	15	4	26.7%	[10.9, 52.0]
2.5-Flash-Lite	0.1	15	8	53.3%	[30.1, 75.2]
2.0-Flash	0.1	15	14	93.3%	[70.2, 98.8]
2.0-Flash	0.9	15	14	93.3%	[70.2, 98.8]
2.5-Flash-Lite	0.9	15	15	100.0%	[79.6, 100.0]

Note: Table sorted by False Positive Rate (FPR) in ascending order. Top-ranking models demonstrate higher robustness against confirmation bias. $95\,\%$ *CI calculated via the Wilson score method.*

have introduced transient latency variations from node rescheduling, despite no evictions being observed during data collection.

External validity: The Online Boutique reference application exhibits lower architectural complexity and traffic than large-scale industrial systems. Moreover, the Istio-based fault injection produces clean, predictable conditions that simplify diagnosis relative to organically occurring failures; in scenario F2, the use of self-descriptive environment variable names (e.g., `EXTRA_LATENCY`) may have further facilitated detection by exposing semantic cues absent in real incidents.

Construct validity: The primary diagnostic metric (SR) relies on a binary classification: either the agent correctly identified both the faulty service and the fault type, or it did not. This strict criterion may underestimate agent capability, as partially correct diagnoses, where the agent identified the correct service but mischaracterized the fault type, were scored as failures. Additionally, the Ground Truth validation was performed by a single SRE engineer.

Conclusion validity: Budgetary constraints limited each configuration to 15 runs per scenario (N = 45 per configuration). To partially mitigate risks associated with low statistical power, a one-tailed binomial test was applied over the 45 aggregated runs, confirming that the best-performing configuration significantly exceeded the random-chance baseline ($p < 0.001$).

8. Conclusion

This study evaluated the feasibility, efficiency, and stability of LLM-based autonomous agents coordinated through the MCP for fault diagnosis in Kubernetes environments. A controlled experiment employing a RCBD quantified the trade-offs between reasoning capacity and operational cost. The findings are summarized as responses to the research questions defined in Section 4.

Regarding **RQ1** (diagnostic efficacy), *Gemini 2.5-Flash* at $T = 0.1$ achieved a $71.1\,\%$ SR, significantly exceeding the random-chance baseline ($p < 0.001$). This result provides empirical evidence that MCP-orchestrated agents possess substantive causal reasoning capability for incident triage in cloud-native environments.

Regarding **RQ2** (economic efficiency), the analysis revealed a *hidden cost of incompetence*: small-parameter models exhibited high token consumption due to redundant reasoning cycles. For instance, *2.0-Flash-Lite* consumed an average of $629{,}967$ tokens despite offering the lowest per-token cost, resulting in total expenditures that surpassed those of more capable models. These findings suggest that per-token pricing alone is an insufficient criterion for model selection in SRE workflows.

Regarding **RQ3** (operational stability), *2.5-Flash-Lite* at $T = 0.1$ exhibited high execution volatility ($CV = 2.17$), whereas *Gemini 2.5-Flash* remained stable across both temperature levels ($CV \leq 0.60$). Additionally, the negative control analysis revealed a non-trivial tension between sensitivity and specificity, with some configurations reaching $93.3\,\%$ and $100\,\%$ false positive rates in healthy systems.

Limitations of this study include the use of synthetic failure scenarios, a scope restricted to three incident types and to the Gemini Flash family, which may affect generalization to heterogeneous production environments. Future work includes extending the evaluation to large-scale real-world incidents under more complex conditions, such as compound faults, noisy telemetry, and partial observability, and comparing additional LLM families.

References

Bagehorn, F., Rios, J., Jha, S., Filepp, R., Shwartz, L., Abe, N., and Yang, X. (2022). A fault injection platform for learning AIOps models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, New York, NY, USA. ACM.

Colaço Júnior, M. (2025). *IA para a Galera Toda: Agentes e Inovação Experimental Sem Código: Guia Prático para Dominar Prompts, Inovar, Criar Experimentos, Artigos e Lucrar com Agentes Inteligentes – Tudo Sem Programação*. Amazon Publishing, Seattle, WA.

Colaço Júnior, M., Cruz, R. F., Araújo, L. V. d., Bliacheriene, A. C., and Nunes, F. d. L. S. (2022). Evaluation of a process for the experimental development of data mining, ai and data science applications aligned with the strategic planning. *JISTEM - Journal of Information Systems and Technology Management*, 19.

Jiang, Z., Huang, J., Yu, G., Chen, Z., Li, Y., Zhong, R., Feng, C., Yang, Y., Yang, Z., and Lyu, M. (2025). *L4: Diagnosing Large-scale LLM Training Failures via Automated Log Analysis*, page 51–63. Association for Computing Machinery, New York, NY, USA.

Li, S., Wei, X., Yuan, J., Wang, X., and Miao, K. (2025). Secure model context protocol for large language models with dual signatures. In *Proceedings of the 20th Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '25, page 1–6, New York, NY, USA. Association for Computing Machinery.

Pei, C., Wang, Z., Liu, F., Li, Z., Liu, Y., He, X., Kang, R., Zhang, T., Chen, J., Li, J., Xie, G., and Pei, D. (2025). Flow-of-action: Sop enhanced llm-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 422–431, New York, NY, USA. Association for Computing Machinery.

Pham, L., Zhang, H., Ha, H., Salim, F., and Zhang, X. (2025). Rcaeval: A benchmark for root cause analysis of microservice systems with telemetry data. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 777–780, New York, NY, USA. Association for Computing Machinery.

Rombaut, B., Masoumzadeh, S., Vasilevski, K., Lin, D., and Hassan, A. E. (2025). Watson: A cognitive observability framework for the reasoning of llm-powered agents.

Severino, A. J. (2018). *Metodologia do trabalho científico*. Cortez, São Paulo, 24 edition.

Shetty, M., Chen, Y., Somashekar, G., Ma, M., Simmhan, Y., Zhang, X., Mace, J., Vandevoorde, D., Las-Casas, P., Gupta, S. M., Nath, S., Bansal, C., and Rajmohan, S. (2024). Building ai agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, SoCC '24, page 99–110, New York, NY, USA. Association for Computing Machinery.

Xiao, X., Gao, C., and Bogner, J. (2025). On the effectiveness of microservices tactics and patterns to reduce energy consumption: An experimental study on trade-offs. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*, page 164–175. IEEE.

Xin, R., Chen, P., and Zhao, Z. (2023). Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. *J. Syst. Softw.*, 203(C).

Xu, H., Wang, X., and Ji, S. (2025a). Reliable and efficient container orchestration of llms via mcp. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, CIKM '25, page 6885–6886, New York, NY, USA. Association for Computing Machinery.

Xu, J., Zhang, Q., Zhong, Z., He, S., Zhang, C., Lin, Q., Pei, D., He, P., Zhang, D., and Zhang, Q. (2025b). OpenRCA: Can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*.

Yang, N., Lyu, G., Ma, M., Lu, Y., Li, Y., Gao, Z., Ye, H., Zhang, J., Chen, T., and Chen, Y. (2025). Iot-mcp: Bridging llms and iot systems through model context protocol. In *Proceedings of the ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '25, page 73–80, New York, NY, USA. Association for Computing Machinery.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models.

Zhang, L., Zhai, Y., Jia, T., Duan, C., He, M., Pan, L., Liu, Z., Ding, B., and Li, Y. (2025). Microremed: Benchmarking llms in microservices remediation.

Zheng, L., Chen, Z., He, J., and Chen, H. (2024). Mulan: Multi-modal causal structure learning and root cause analysis for microservice systems. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 4107–4116, New York, NY, USA. Association for Computing Machinery.

Zota, R. D., Bărbulescu, C., and Constantinescu, R. (2025). A practical approach to defining a framework for developing an agentic AIOps system. *Electronics (Basel)*, 14(9):1775.