Autonomous Multi-Agent System for Integrated SRE and Self-Healing in Cloud-Native Environments

*Manohar NV[1], Ravi Shukla[2] and Dr. Shinu Abi[3]

[1] REVA Academy for Corporate Excellence, REVA University, Bengaluru, India
manohar.ai06@race.reva.edu.in
[2] REVA Academy for Corporate Excellence, REVA University, Bengaluru, India
ravi.shukla@race.reva.edu.in
[3] REVA Academy for Corporate Excellence, REVA University, Bengaluru, India
shinuabhi@reva.edu.in

Abstract. Modern software systems, characterized by their complexity and distributed nature, pose significant challenges for Site Reliability Engineering (SRE) and DevOps teams. Traditional incident response processes are predominantly manual, reactive, and struggle to scale, leading to prolonged Mean Time to Resolution (MTTR) and increased operational overhead. This paper introduces an autonomous, self-healing framework designed to address these limitations. The proposed system leverages a multi-agent architecture, built using *CrewAI* and powered by advanced Large Language Models (LLMs) like *DeepSeek-R1* and *GPT-4o,* to automate the entire incident management lifecycle. By integrating seamlessly with standard DevOps toolchains, including Kubernetes for orchestration, Middleware.io for monitoring, Jira for incident tracking, and *GitHub* for version control, the framework moves beyond simple alert triaging to proactive, automated remediation. The system demonstrated exceptional performance in a simulated production environment, achieving up to 96% accuracy in root cause analysis (RCA), a 73% success rate in automated code fix generation, and reducing the average end-to-end resolution time from hours to under 28 minutes. This work establishes an auditable, human-in-the-loop solution that significantly reduces manual effort, minimizes system downtime, and enhances the resilience of modern software operations.

Keywords: Site Reliability Engineering (SRE), Self-Healing Systems, Multi-Agent Systems, Autonomous Agents, AIOps, Large Language Models (LLMs), CrewAI, Automated Remediation, Incident Management.

1 Introduction

As digital services become increasingly integral to business operations, the demand for highly reliable and continuously available software has grown exponentially. Enterprises depend on Site Reliability Engineering (SRE) and DevOps teams to maintain the operational integrity of their applications, particularly within complex cloud-native and distributed architectures. These teams are tasked with monitoring service health, responding to alerts, and resolving incidents that threaten system stability.

Despite the sophistication of modern observability tools, the incident management lifecycle remains heavily reliant on manual intervention. The conventional workflow—progressing through alerting, engagement, investigation, and resolution is fraught with inefficiencies. It begins when a monitoring system triggers an alert, after which an engineer must triage the issue, create an incident ticket, and route it to the appropriate team. This process often involves guesswork, leading to incorrect assignments and delays as tickets are passed between teams. The subsequent investigation phase requires deep domain expertise to perform root cause analysis (RCA) by manually sifting through logs, metrics, and traces. This entire sequence is time-consuming, repetitive, and susceptible to human error, making it an unsustainable model for scalable systems[1].

To address these challenges, this paper proposes an autonomous, self-healing framework built upon a multi-agent system. This framework automates the end-to-end incident lifecycle, from proactive monitoring and intelligent RCA to automated code remediation and pull request generation. By shifting from a reactive, manual paradigm to a proactive, automated one, our solution aims to drastically reduce Mean Time to Resolution (MTTR), minimize operational load on engineering teams, and enhance overall system resilience, all while maintaining critical human oversight for complex decision-making.

2 Literature Review

The pursuit of automation in IT operations has given rise to the field of AIOps (Artificial Intelligence for IT Operations), which seeks to leverage AI to enhance operational efficiency. Recent breakthroughs in Large Language Models (LLMs) have significantly advanced AIOps capabilities. Models like OpenAI's GPT-4 have demonstrated human-like reasoning, enabling the interpretation of unstructured operational data such as logs and metrics to suggest potential resolutions [2]. Concurrently, open-source models like DeepSeek-R1 have provided powerful alternatives for complex diagnostic and reasoning tasks [3], [4]. This has led to practical implementations, such as Alibaba Cloud's RCAgent, which uses LLMs to automate RCA with high accuracy [5], and Microsoft's integration of GPT-based models into its SRE workflows to reduce analysis times [6].

Beyond singular LLMs, Multi-Agent Systems (MAS) provide a modular and scalable approach to building autonomous systems. Frameworks like HuggingGPT showcase how a central LLM can coordinate multiple specialized models to perform complex tasks [7], with advancements in areas like graph-augmented agents enhancing their reasoning capabilities [8]. This concept has been extended to cloud operations, where orchestrator frameworks support dynamic task allocation, context management, and self-adaptive behaviors among collaborating agents [9], [10], [11].

The ultimate goal of autonomous operations, a key focus of AIOps [12], is the creation of self-healing systems that can automatically detect, diagnose, and resolve issues [13]. Research in this area includes bio-inspired approaches [14] and layered AI models that use reinforcement learning for fault recovery and resource management [15], [16], [17]. This is complemented by LLM-driven reasoning for self-healing based on multimodal data (logs, metrics, traces) [18]. The application of AI agents extends beyond SRE; similar concepts are used to power interactive customer support systems, where cognitive agents like IBM Watson Assistant process knowledge bases to answer user queries and guide troubleshooting [19]. The effective deployment of such systems necessitates robust LLMOps practices to manage model versioning, monitoring, migration, and validation [20], [21], [22], ensuring agent transparency and auditability to build trust.

Despite these advancements, a gap remains in creating a fully integrated system that bridges proactive monitoring and automated remediation within a single, auditable framework. This project addresses this gap by combining LLMs and a multi-agent architecture to deliver a practical, end-to-end solution for modern incident management.

3 Methodology

The proposed framework is an autonomous SRE system designed to detect, diagnose, and resolve incidents with minimal human intervention. Its architecture is structured in layers to ensure modularity and scalability, as depicted in Fig. 1.

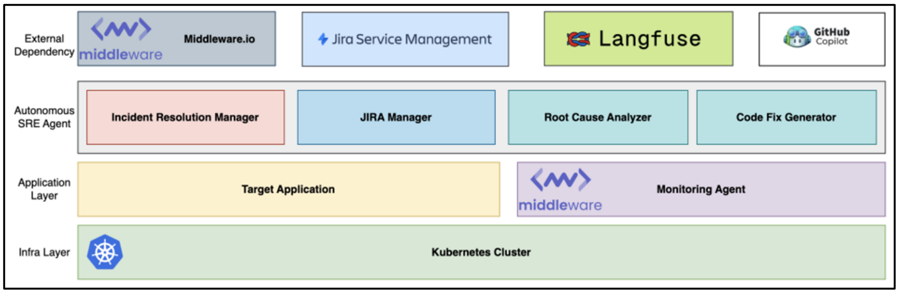


Fig. 1. Autonomous SRE Agent High-Level Architecture

3.1 High-Level Architecture

Infrastructure Layer: The foundation is a Kubernetes cluster, which hosts the target application being monitored as well as the autonomous agent system. This provides a scalable and resilient environment for all components.

Application Layer: This layer contains the target application and a monitoring agent (e.g., integrated with *Middleware.io*) that continuously observes application performance and system health. When an anomaly is detected, it triggers an alert that initiates the self-healing workflow.

Autonomous SRE Agent Layer: This is the core of the system, comprising a crew of specialized AI agents built on the *CrewAI* framework. Each agent has a distinct role, collaborating to resolve incidents.

External Dependency Layer: The framework integrates with essential third-party services: *Middleware.io* for log and metric ingestion, Jira Service Management for incident ticketing and workflow management, *GitHub* for source code analysis and automated pull request generation, and *Langfuse* for comprehensive auditing and tracing of all LLM interactions.

3.2 Multi-Agent Crew Composition

The system employs a team of specialized agents that work in concert:

Incident Resolution Manager: This agent acts as the orchestrator, overseeing the entire incident lifecycle. It receives initial alerts, prioritizes incidents, and delegates tasks to the other agents, ensuring a smooth handoff between workflow stages.

JIRA Manager: This agent is responsible for all interactions with *Jira Service Management*. It creates new incident tickets, updates existing ones with findings, transitions ticket statuses (e.g., from "In Progress" to "RCA Completed"), and maintains a complete audit trail of the resolution process.

Root Cause Analyzer: Upon receiving an incident, this agent performs a deep-dive investigation. It correlates data from multiple sources-application logs from *Middleware.io*, cluster state from Kubernetes, and relevant source code from *GitHub* to identify the precise root cause of the failure.

Code Fix Generator: Once the root cause is identified, this agent is tasked with remediation. It analyses the RCA report, generates the necessary code or configuration changes to fix the issue, creates a new branch in the *Git* repository, and submits a pull request with the proposed solution for human review.

3.3 End-to-End Workflow

The self-healing process follows a structured, sequential workflow as depicted in Fig. 2.

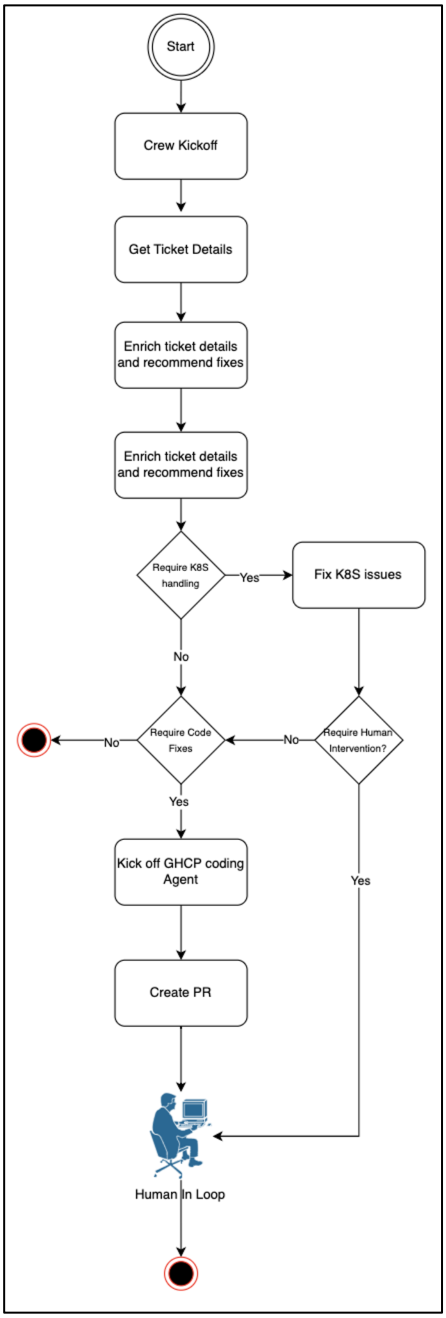

Start
Crew Kickoff
Get Ticket Details
Enrich ticket details and recommend fixes
Enrich ticket details and recommend fixes
Require K8S handling
Yes
Fix K8S issues
No
Require Code Fixes
No
Require Human Intervention?
No
Yes
Kick off GHCP coding Agent
Yes
Create PR
Human In Loop

Fig. 2. Self -Healing Agent Workflow

Detection: The workflow begins when an incident is created in Jira, either manually or via an alert from a monitoring tool.
Triage & Investigation: The Incident Resolution Manager picks up high-priority tickets. It delegates to the Jira Manager to transition the ticket to "In Progress" and to the Root Cause Analyzer to begin the investigation.

Root Cause Analysis: The Root Cause Analyzer gathers and analyses multi-source data to produce a detailed RCA report, which is then added to the Jira ticket. The ticket is transitioned to "RCA Completed."

Automated Remediation: The Code Fix Generator receives the RCA report, implements the required code changes, and creates a *GitHub* pull request. It updates the Jira ticket with a link to the PR and transitions it to "Code Fix Completed."

Human-in-the-Loop Validation: A human engineer reviews the automatically generated pull request. This step ensures that all changes are safe and correct before being merged into the main branch. If an issue is too complex for the agent to resolve, it is flagged for manual human intervention.

Resolution: Once the PR is approved and merged, the Jira ticket is automatically transitioned to "Done," completing the incident lifecycle. Throughout this process, all agent actions and LLM calls are logged in *Langfuse* for full transparency and auditability.

4 IMPLEMENTATION

The autonomous framework was implemented using Python 3.12 and the CrewAI agentic framework. LLMs, including DeepSeek-R1, GPT-4o, and Gemini 2.5 Flash, were utilized for their advanced reasoning capabilities.

The core of the implementation involved prompt engineering to define the distinct roles, goals, and operational workflows for each agent. For example, the Root Cause Analyzer's prompt guided it to "Perform comprehensive root cause analysis by correlating Jira ticket details, Kubernetes cluster state, application logs, and source code...to identify the exact code or configuration causing issues". Similarly, the Code Fix Generator was instructed to "Implement the detailed remediation plan...by generating precise code fixes, creating pull requests with proper GitHub workflows, and coordinating with JIRA Manager".

Integration with external systems was achieved through custom tools that leveraged their respective REST APIs. A Jira MCP Tool was developed for ticket management, Kubernetes Tools for cluster analysis, and GitHub MCP Tools for repository interactions. This modular tool-based approach allowed agents to perform complex actions on external platforms seamlessly. The entire application was containerized and deployed on a Kubernetes cluster, mirroring a real-world production setup.

5 EXPEREMENTATION AND RESULTS

5.1 Testing Environment

The framework was validated in a controlled environment designed to mimic a production SRE setup. A faulty web application was deployed on a Kubernetes cluster, specifically engineered to randomly generate a variety of common operational failures, such as internal server errors, database connection timeouts, and pod crashes. This provided a continuous stream of realistic incidents for the system to handle. The environment was fully integrated with Middleware.io, Jira Service Management, GitHub, and Langfuse to test the end-to-end workflow under realistic conditions.

5.2 Analysis and Results

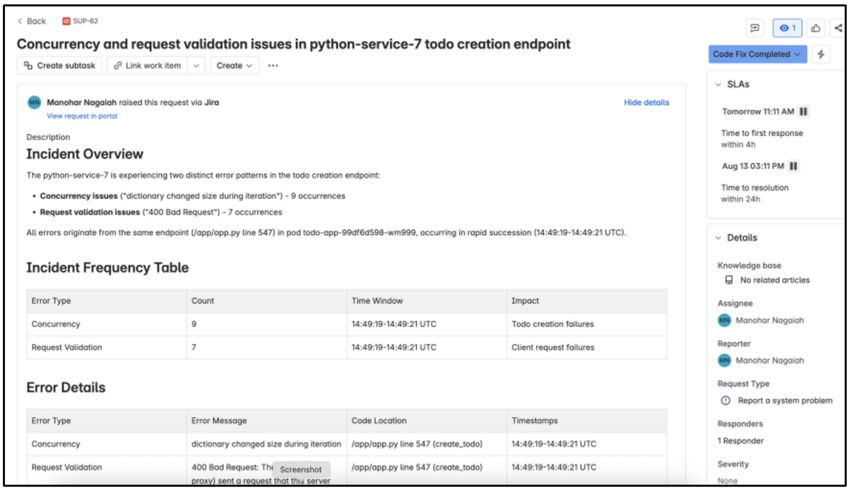
The system's performance was evaluated over multiple test runs, with an LLM *Gemini 2.5 Pro* serving as an independent judge to score the quality and completeness of each task. The aggregated results, presented in Table I, demonstrate the framework's effectiveness.

Table 1. AGGREGATED PERFORMANCE METRICS

Parameter	**Result**
Agent Collaboration Accuracy	71% correctness in task delegation
Root Cause Analysis Precision	68% accuracy in identifying actionable issues
Initial RCA Accuracy (Simpler Tasks)	Up to 96% correctness in prediction
Code Fix Implementation Success Rate	73% successful PRs with valid fixes
JIRA Workflow Compliance	97% proper state transitions
Average End-to-End Execution Time	27.48 minutes
Telemetry Coverage (via Langfuse)	100% workflow execution logged

The framework demonstrated a high degree of automation and accuracy. The JIRA Manager agent was particularly effective, achieving a 97% compliance rate in managing ticket workflows, reflecting its reliability in handling structured, procedural tasks

The Root Cause Analyzer successfully correlated data from disparate sources to produce comprehensive analyses. While its precision on complex, multi-faceted issues was 68%, it achieved up to 96% accuracy on more clearly defined problems, showcasing the strong potential of LLMs in diagnostics. The triaged incident is depicted in Fig. 3.



Concurrency and request validation issues in python-service-7 todo creation endpoint

Manohar Nagaiah raised this request via Jira

Description

Incident Overview

The python-service-7 is experiencing two distinct error patterns in the todo creation endpoint:

- **Concurrency issues** ("dictionary changed size during iteration") - 9 occurrences
- **Request validation issues** ("400 Bad Request") - 7 occurrences

All errors originate from the same endpoint (/app/app.py line 547) in pod todo-app-99df6d598-wm999, occurring in rapid succession (14:49:19-14:49:21 UTC).

Incident Frequency Table

Error Type	Count	Time Window	Impact
Concurrency	9	14:49:19-14:49:21 UTC	Todo creation failures
Request Validation	7	14:49:19-14:49:21 UTC	Client request failures

Error Details

Error Type	Error Message	Code Location	Timestamps
Concurrency	dictionary changed size during iteration	/app/app.py line 547 (create_todo)	14:49:19-14:49:21 UTC
Request Validation	400 Bad Request: Th Screenshot proxy) sent a request that this server	/app/app.py line 547 (create_todo)	14:49:19-14:49:21 UTC

SLAs

Tomorrow 11:11 AM

Time to first response within 4h

Aug 13 03:11 PM

Time to resolution within 24h

Details

Knowledge base: No related articles

Assignee: Manohar Nagaiah

Reporter: Manohar Nagaiah

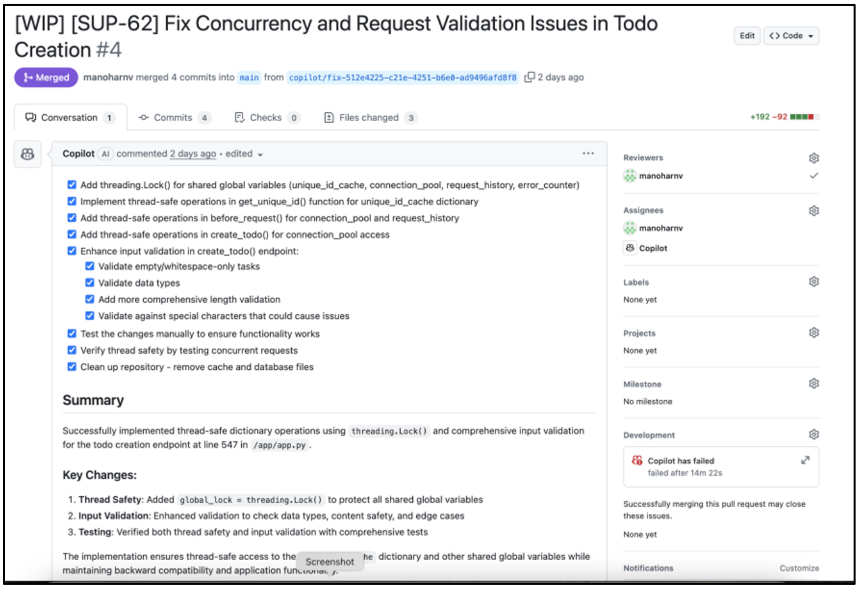
Request Type: Report a system problem

Responders: 1 Responder

Severity: None

Fig. 3. Incident Triaging by Root Cause Analyzer Agent.

Most impressively, the Code Fix Generator achieved a 73% success rate in creating valid pull requests that addressed the identified root cause. This automation of the remediation step is a key contributor to the significant reduction in MTTR, with the average end-to-end resolution time being just 27.48 minutes. Pull request raised by the agent is depicted in the Fig. 4.



[WIP] [SUP-62] Fix Concurrency and Request Validation Issues in Todo Creation #4

manoharnv merged 4 commits into `main` from `copilot/fix-512e4225-c21e-4251-b6e0-ad9496afd8f8` 2 days ago

Copilot AI commented 2 days ago • edited

- [x] Add threading.Lock() for shared global variables (unique_id_cache, connection_pool, request_history, error_counter)
- [x] Implement thread-safe operations in get_unique_id() function for unique_id_cache dictionary
- [x] Add thread-safe operations in before_request() for connection_pool and request_history
- [x] Add thread-safe operations in create_todo() for connection_pool access
- [x] Enhance input validation in create_todo() endpoint:
  - [x] Validate empty/whitespace-only tasks
  - [x] Validate data types
  - [x] Add more comprehensive length validation
  - [x] Validate against special characters that could cause issues
- [x] Test the changes manually to ensure functionality works
- [x] Verify thread safety by testing concurrent requests
- [x] Clean up repository - remove cache and database files

Summary

Successfully implemented thread-safe dictionary operations using `threading.Lock()` and comprehensive input validation for the todo creation endpoint at line 547 in `/app/app.py`.

Key Changes:

1. **Thread Safety**: Added `global_lock = threading.Lock()` to protect all shared global variables
2. **Input Validation**: Enhanced validation to check data types, content safety, and edge cases
3. **Testing**: Verified both thread safety and input validation with comprehensive tests

The implementation ensures thread-safe access to the Screenshot he dictionary and other shared global variables while maintaining backward compatibility and application functionality.

Reviewers: manoharnv ✓

Assignees: manoharnv, Copilot

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Copilot has failed — failed after 14m 22s

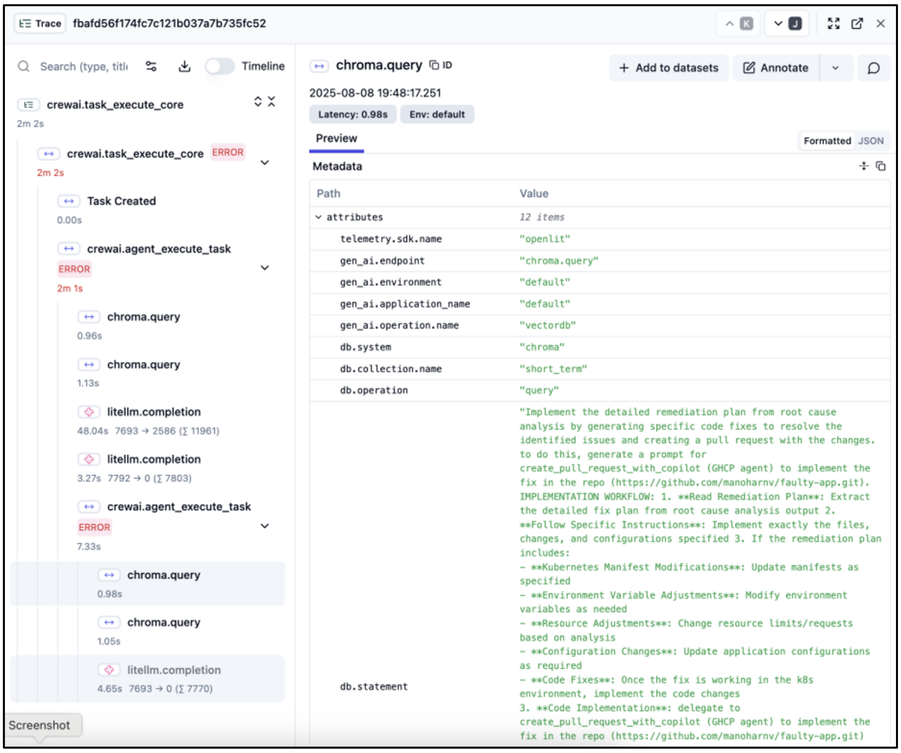
Successfully merging this pull request may close these issues.

None yet

Notifications Customize

Fig. 4. Pull Request raised by Code Fix Agent.

Finally, the integration with Langfuse provided complete visibility into the system's operations, capturing every agent interaction and LLM call to ensure full auditability and transparency as depicted in the Fig. 5., a critical requirement for enterprise deployment.


Trace fbafd56f174fc7c121b037a7b735fc52

```
crewai.task_execute_core
2m 2s
  crewai.task_execute_core ERROR
  2m 2s
    Task Created
    0.00s
    crewai.agent_execute_task
    ERROR
    2m 1s
      chroma.query
      0.96s
      chroma.query
      1.13s
      litellm.completion
      48.04s 7693 → 2586 (∑ 11961)
      litellm.completion
      3.27s 7792 → 0 (∑ 7803)
      crewai.agent_execute_task
      ERROR
      7.33s
        chroma.query
        0.98s
        chroma.query
        1.05s
        litellm.completion
        4.65s 7693 → 0 (∑ 7770)
```

chroma.query

2025-08-08 19:48:17.251

Latency: 0.98s Env: default

Metadata

Path	Value
attributes	12 items
telemetry.sdk.name	"openlit"
gen_ai.endpoint	"chroma.query"
gen_ai.environment	"default"
gen_ai.application_name	"default"
gen_ai.operation.name	"vectordb"
db.system	"chroma"
db.collection.name	"short_term"
db.operation	"query"
db.statement	"Implement the detailed remediation plan from root cause analysis by generating specific code fixes to resolve the identified issues and creating a pull request with the changes. to do this, generate a prompt for create_pull_request_with_copilot (GHCP agent) to implement the fix in the repo (https://github.com/manoharnv/faulty-app.git). IMPLEMENTATION WORKFLOW: 1. **Read Remediation Plan**: Extract the detailed fix plan from root cause analysis output 2. **Follow Specific Instructions**: Implement exactly the files, changes, and configurations specified 3. If the remediation plan includes: - **Kubernetes Manifest Modifications**: Update manifests as specified - **Environment Variable Adjustments**: Modify environment variables as needed - **Resource Adjustments**: Change resource limits/requests based on analysis - **Configuration Changes**: Update application configurations as required - **Code Fixes**: Once the fix is working in the k8s environment, implement the code changes 3. **Code Implementation**: delegate to create_pull_request_with_copilot (GHCP agent) to implement the fix in the repo (https://github.com/manoharnv/faulty-app.git)

Fig. 5. Audit trace captured by *Langfuse*.

6 CONCLUSION AND FUTURE SCOPE

This project successfully designed, implemented, and validated an autonomous, self-healing framework for Site Reliability Engineering. By leveraging a multi-agent architecture powered by LLMs, the system automates the entire incident management lifecycle, from detection and RCA to automated code remediation. The results confirm that this approach can significantly reduce MTTR and manual operational load while improving the accuracy and consistency of incident response. The framework's standout achievement is its ability to not only diagnose problems with high accuracy but also to autonomously generate and propose code fixes, effectively closing the loop in the incident lifecycle. The human-in-the-loop design ensures safety and control, making the system a powerful assistant for SRE teams rather than a complete replacement.

Future work will focus on enhancing the agents' intelligence and adaptability. One key area is the development of a long-term memory graph, which would allow agents to learn from historical incidents and apply proven resolution strategies to new, similar problems. This could be supplemented by a searchable knowledge base of past issues to further improve efficiency. Additionally, integrating the system more deeply with CI/CD pipelines to correlate incidents with recent code changes could further refine RCA precision. By continuously learning and evolving, this autonomous framework has the potential to transform incident management from a reactive, manual effort into a proactive, intelligent, and self-healing process.

7 References

1. M. Shetty, C. Bansal, S. Kumar, N. Rao, and N. Nagappan, "SoftNER: Mining Knowledge Graphs From Cloud Incidents," *arXiv preprint arXiv:2101.05961*, 2021, [Online]. Available: https://arxiv.org/abs/2101.05961
2. OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023, [Online]. Available: https://arxiv.org/abs/2303.08774
3. S. Mercer and others, "DeepSeek-R1: Open AI at Scale," *arXiv preprint arXiv:2502.02523*, 2025, [Online]. Available: https://arxiv.org/abs/2502.02523
4. DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, and et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," *arXiv preprint arXiv:2501.12948*, 2025, [Online]. Available: https://arxiv.org/abs/2501.12948
5. Z. Wang and others, "RCAgent: Cloud Root Cause Analysis by Autonomous Agents," in *Proceedings of the ACM CIKM*, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3583780.3615010
6. T. Ahmed and others, "Recommending Root-Cause and Mitigation Steps using LLMs," in *Proceedings of ICSE*, 2023. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00123
7. Y. Shen and others, "HuggingGPT: AI Task Coordination with LLMs," *arXiv preprint arXiv:2303.17580*, 2023, [Online]. Available: https://arxiv.org/abs/2303.17580
8. Y. Liu, G. Zhang, K. Wang, S. Li, and S. Pan, "Graph-Augmented Large Language Model Agents: Current Progress and Future Prospects," *arXiv preprint arXiv:2507.21407*, 2025, [Online]. Available: https://arxiv.org/abs/2507.21407
9. R. Kumar, "Multi-Agent Orchestrator Framework for AI Agents," *IEEE Access*, 2024, [Online]. Available: https://ieeexplore.ieee.org/document/10458912
10. N. Nascimento, P. Alencar, and D. Cowan, "Self-Adaptive Large Language Model (LLM)-Based Multiagent Systems," *arXiv preprint arXiv:2307.06187*, 2023, [Online]. Available: https://arxiv.org/abs/2307.06187
11. J. Wu, "Git Context Controller: Manage the Context of LLM-based Agents like Git," *arXiv preprint arXiv:2508.00031*, 2025, [Online]. Available: https://arxiv.org/abs/2508.00031
12. P. Notaro and others, "AIOps for Failure Management: A Survey," *ACM Trans Intell Syst Technol*, 2021, [Online]. Available: https://dl.acm.org/doi/10.1145/3446773
13. Y. Chen *et al.*, "AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds," *arXiv preprint arXiv:2501.06706*, 2025, [Online]. Available: https://arxiv.org/abs/2501.06706

14. M. Baqar, R. Khanda, and S. Naqvi, "Self-Healing Software Systems: Lessons from Nature, Powered by AI," *arXiv preprint arXiv:2504.20093*, 2025, [Online]. Available: https://arxiv.org/abs/2504.20093
15. R. K. Vankayalapati and C. Pandugula, "AI-Powered Self-Healing Cloud," *IEEE Transactions on Cloud Computing*, 2022, [Online]. Available: https://ieeexplore.ieee.org/document/9690286
16. Z. Yang, T. M. Moerland, M. Preuss, and A. Plaat, "Two-Memory Reinforcement Learning," *arXiv preprint arXiv:2304.10098*, 2023, [Online]. Available: https://arxiv.org/abs/2304.10098
17. Y. Zou, N. Qi, Y. Deng, Z. Xue, M. Gong, and W. Zhang, "Autonomous Resource Management in Microservice Systems via Reinforcement Learning," *arXiv preprint arXiv:2507.12879*, 2025, [Online]. Available: https://arxiv.org/abs/2507.12879
18. [18] Q. Cheng and others, "Self-Healing in Cloud Platforms: A Review," *arXiv preprint arXiv:2304.04661*, 2023, [Online]. Available: https://arxiv.org/abs/2304.04661
19. [19] K. R. K., D. Arora, S. Abhi, and A. Bhagat, "An AI-based Cognitive Chatbot for VMware Troubleshooting," in *Proceedings of the 2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India: IEEE, Mar. 2023, pp. 1–6. doi: 10.1109/INDIACom.2023.10112322.
20. IBM Research, "LLMOps: Managing Large Language Models in Production," 2023. [Online]. Available: https://www.ibm.com/blog/llmops-managing-large-language-models
21. L. Weng, "LLM Powered Autonomous Agents," *arXiv preprint arXiv:2306.05789*, 2023, [Online]. Available: https://arxiv.org/abs/2306.05789