Agent Iris: Using Artificial Intelligence for Incident Analysis in Production Environments at Large-Scale Services

Rita Oliveira
iFood
Carnaubal, Brazil
rita.oliveira@ifood.com.br

Danilo Clemente
iFood
Lavras, Brazil
danilo.clemente@ifood.com.br

Heitor Costa
Department of Computer Science,
Federal University of Lavras
Lavras, Brazil
heitor@ufla.br

Abstract

Organizations operating large-scale microservice architectures face significant challenges in diagnosing production incidents due to the complexity of distributed request flows. This paper presents Agent Iris, an intelligent assistant deployed at iFood to serve as a first-response diagnostic layer for incident analysis. The agent combines Chain-of-Thought reasoning with Retrieval-Augmented Generation grounded in Domain-Driven Design documentation, and integrates with Datadog via the Model Context Protocol for real-time observability. An empirical evaluation was conducted using five commercial language models, with previously resolved incidents serving as oracles. Results show that GPT-4.1 provides the optimal balance between response latency (8 seconds on average) and diagnostic accuracy, reducing the initial investigation time from approximately 5–15 minutes to seconds. This paper discusses the practical lessons learned from deploying an LLM-based agent in a production SRE context, including model selection trade-offs, documentation strategies, and a roadmap toward proactive incident resolution.

Keywords

AI Agents, Language Models, Observability, Datadog, Incidents, Microservices

1 Introduction

In large-scale technology companies such as iFood[1], it is common for different business workflows to operate across dozens of microservices employing complex architectural patterns such as event-driven communication, asynchronous messaging, and distributed transactions [7, 11]. These patterns, while enabling scalability and resilience, significantly increase the difficulty of tracing end-to-end request flows and expand the area of knowledge required to diagnose failures [6].

When a critical incident occurs, development and Site Reliability Engineering (SRE) teams face a high cognitive load: they must navigate multiple observability dashboards, correlate logs across services, and distinguish between infrastructure failures and business-logic errors—a process that routinely consumes 5 to 15 minutes per incident [3, 8]. This paper presents Agent Iris, an intelligent assistant developed at iFood to serve as a first-response diagnostic layer for production incidents. The main contributions of this work are:

(1) An empirical evaluation of five commercial language models applied to real incident scenarios, using previously resolved cases as oracles to measure diagnostic accuracy and response latency;
(2) A discussion of practical lessons learned from deploying an LLM-based agent in a production SRE context, including model selection trade-offs, documentation strategies, and tool management decisions.

The remainder of this paper is organized as follows. Section 2 describes the Agent Iris architecture. Section 3 presents the experimental evaluation. Section 4 discusses the results and lessons learned. Section 5 concludes with future directions.

2 Agent Iris

We designed the Agent Iris to investigate errors in iFood's payment flow. Its primary objective is to rapidly locate logs, recent deployments, service metrics, and other observability signals that may indicate the origin of a problem, providing developers with a preliminary diagnosis and actionable suggestions.

2.1 Platform and Tools

We deployed the agent on the Toqan platform[2], an enterprise artificial intelligence platform developed by Prosus[3] in partnership with iFood for creating and orchestrating intelligent agents. Toqan provides a unified interface for deploying agents equipped with custom tools across diverse infrastructure integrations and enables seamless switching between foundation models based on latency and reasoning requirements.

The agent utilizes a specific set of tools, whether built manually, obtained via the Model Context Protocol (MCP)[4] [1], or embedded natively within Toqan. Datadog[5] was not actively chosen over alternatives; it was used because it is the established standard for logs, metrics, and Kubernetes[6] cluster monitoring at iFood. Additionally, using Toqan's integrated LLMs provided the fastest time-to-value for this initiative, whereas fine-tuning or local models may be considered as the project scales. Each tool is essentially a sequence of instructions that teaches the model how to perform a specific action, such as querying an API (*Application Programming Interface*), accessing a dashboard, or retrieving a document. The primary Datadog tools available via MCP are:

- `datadog-get-dashboard` retrieves dashboard metadata containing current metric query structures;

[1] https://www.ifood.com.br/

[2] https://work.toqan.ai/welcome
[3] https://www.prosus.com/
[4] https://modelcontextprotocol.io
[5] https://www.datadoghq.com/
[6] https://kubernetes.io/pt-br/

- `datadog-query-metrics` executes metric queries to validate service health;
- `datadog-get-metrics-error-rate` analyzes HTTP (HyperText Transfer Protocol) error rates for specific services;
- `datadog-get-consumer-lag` measures Kafka consumer lag with emergency log filtering;
- `datadog-payment-conversion` validates payment service conversion metrics.

2.2 Reasoning and Knowledge Base

The agent leverages a Chain of Thought [13] approach to establish clear decision pathways, guiding the model through logical conditions (e.g., "if the context indicates an infrastructure issue, investigate metrics; if it indicates a business error, consult domain documentation"). Currently, this reasoning is defined via explicit prompts that instruct the agent exactly where to look and which skills to use, rather than relying on autonomous self-improvement loops. This structured reasoning is integrated with Retrieval-Augmented Generation (RAG) [10] via Confluence [2], which provides the agent with technical details on specific solutions and operational procedures.

The documentation stored in Confluence follows Domain-Driven Design (DDD) principles [6] and is written in a ubiquitous language. Specifically, applying the Ubiquitous Language principle aligned the terminology used by technical systems (e.g., microservice names, metric labels) with the business domain. This explicit mapping ensures that both technical engineers and business stakeholders can read, maintain, and extend the knowledge base without ambiguity. The agent can receive instructions ranging from highly technical directives to high-level business descriptions, and in both cases, understand the appropriate course of action. This documentation also serves as a gating mechanism: if no documented context matches the incoming request, the agent explicitly informs the user rather than attempting to infer a solution.

2.3 System Architecture

Figure 1 illustrates the system architecture. The numbered labels indicate the processing order: (1) the user submits an incident context; (2) the Toqan platform receives the query; (3) the `confluence-get-content` tool is used to retrieve instructions from the Confluence API; (4) the retrieved Confluence documentation, containing workflows and skills, feeds into (5) the Agent Workstation, which applies Chain-of-Thought conditional logic to select the appropriate diagnostic skill; (6) the Skill Orchestration module invokes the relevant Datadog tools via MCP and aggregates the findings; and (7) the final analysis result, containing root causes and suggestions, is returned to the user.

2.4 Observability and Dashboard-Driven Queries

Datadog serves as the agent's primary observability source. The agent accesses dashboards and metrics exported from relevant services, enabling identification of deployments, external service latency, throughput anomalies, critical-severity logs, and messaging metrics from Apache Kafka and Amazon SQS.

Rather than embedding hard-coded metric queries in tool calls, Agent Iris employs an indirection pattern: the agent first retrieves a dashboard containing the current query structure, then executes the metric query using parameters defined in the dashboard. This approach provides query resilience (when metric names or dashboard widgets change, the underlying query structure adapts automatically) and reduces brittleness compared to earlier implementations where hard-coded queries failed whenever metric definitions evolved.

2.5 Execution Safety and Loop Prevention

A critical challenge in agent design is preventing loops, a failure mode in which agents repeatedly invoke tools with modified parameters when expected outputs are not obtained, consuming significant compute resources [9]. To address this, Agent Iris employs a modular documentation approach with explicit break conditions. The agent's initial prompt is minimal and directive, instructing it to retrieve its full operational instructions from Confluence before taking any action. It enforces a two-step process that prevents uncontrolled exploration.

Additionally, tool instructions are embedded directly in the Toqan platform rather than in the Confluence knowledge base. This architectural decision eliminates an intermediate retrieval step, reduces hallucination by treating tool specifications as ground truth within the agent's context [12], and ensures that the agent accurately maps incident context to the correct action [5].

2.6 End-to-End Example: Messaging Lag

To illustrate the workflow, consider a scenario in which an Amazon SQS queue or a Kafka topic experiences processing delay. Datadog detects the anomaly and triggers a Slack webhook alert, which Toqan intercepts (e.g., "payment-service-x has lag in Kafka topic xxx"). The agent processes this context, determines it requires log analysis, and uses the Datadog query tools to search for exceptions or errors specifically within the `payment-service-x` logs. Even if a user manually typed this error string into the agent's chat interface, the agent would follow the identical autonomous reasoning path to identify the root cause.

3 Experimental Evaluation

The selection of the underlying language model was a critical design decision for Agent Iris. Rather than treating this as a standard benchmarking task, the evaluation was designed to identify the model best suited for operational incident response, where both response latency and diagnostic accuracy are essential.

3.1 Methodology

Five commercial language models were evaluated: Claude Sonnet 4.5[7], GPT-4.1[8], GPT-4o[9], GPT-5 Max[10], and GPT-5 Min[11]. We tested each model against a set of previously resolved production incidents with known root causes. The selected incidents represented

[7] https://www.anthropic.com/news/claude-sonnet-4-5
[8] https://openai.com/index/gpt-4-1/
[9] https://openai.com/index/hello-gpt-4o
[10] https://openai.com/pt-BR/gpt-5/
[11] https://openai.com/pt-BR/gpt-5/

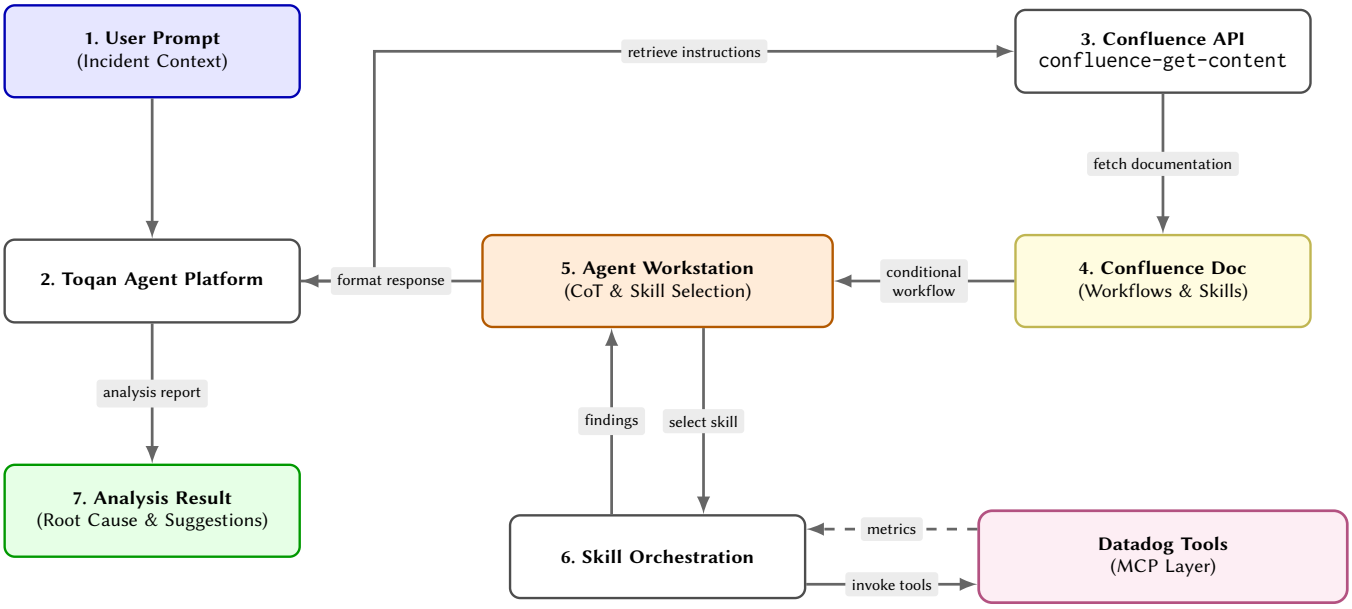

Figure 1: Architectural workflow of Agent Iris. Numbered labels indicate the processing order during the incident analysis lifecycle.

complex scenarios that typically require 5 to 15 minutes of manual investigation by an experienced engineer. For each model, we recorded the following metrics: average response latency, average number of tool-call steps, and adherence to the Confluence documentation retrieval gate.

3.2 Model Comparison

The evaluation revealed three distinct behavioral profiles across the tested models. Table 1 summarizes the comparative operational metrics.

Table 1: Operational Comparison of Evaluated Models

Model	Avg. Latency (s)	Avg. Steps	Operational Behavior
Claude Sonnet 4.5	35	10	Overthinking
GPT-4.1	8	6	Balanced (Selected)
GPT-4o	5	4	Unreliable
GPT-5 Max	30	10	Overthinking
GPT-5 Min	4	3	Unreliable

Overthinking models (Claude Sonnet 4.5 and GPT-5 Max) averaged 30 to 35 seconds per response, executing up to 10 sequential tool calls and attempting to retrieve an excessive amount of auxiliary documentation. While 30 seconds is significantly faster than the 5 to 15 minutes required for manual diagnosis, a faster model can still provide a sufficiently accurate and actionable diagnosis. Although slower models can generate a more exhaustive response, both are valid for resolving the incident. Thus, prioritizing the faster model minimizes time-to-resolution in critical scenarios.

Unreliable models (GPT-4o and GPT-5 Min) responded in under 5 seconds but showed a strong tendency to exceed operational safety bounds. They frequently skipped the Confluence documentation retrieval step or failed when parsing nested JSON structures returned by Datadog API tools, resulting in fast but operationally unusable guidance [4].

Balanced model (GPT-4.1) achieved an average response time of 8 seconds with 6 tool-call steps. It consistently adhered to the documentation retrieval gate, accurately mapped incident context to the correct Datadog tools, and delivered concise, actionable diagnostic suggestions. We selected this model for production deployment.

4 Results and Lessons Learned

The deployment and evaluation of Agent Iris yielded several practical insights relevant to the broader adoption of LLM-based agents in operational contexts.

Simpler models can outperform complex ones in operational settings. GPT-4.1 outperformed models with higher parameter counts and more advanced reasoning capabilities. In incident response, speed and adherence to predefined workflows are more valuable than deep, exploratory reasoning. This finding challenges the assumption that the most capable model is always the best choice for production deployment.

Grounding RAG in ubiquitous language proved highly beneficial. Documentation written following DDD principles [6] and using a ubiquitous language demonstrated significant utility, enabling the agent to bridge low-level technical infrastructure knowledge and high-level business domain logic. This approach reduced semantic ambiguity during query resolution and enabled both engineers and business stakeholders to collaboratively maintain the knowledge base without confusing the language model.

Tool embedding location matters. Placing tool instructions directly in the agent platform (Toqan), rather than in the RAG knowledge source (Confluence), eliminated an intermediate retrieval step and reduced hallucinations to near zero. The trade-off is that tool updates require publishing a new agent version rather than simply editing a Confluence page.

Modular documentation prevents agent loops. Separating the agent's identity prompt from its operational instructions and requiring an explicit retrieval step before any action created hard behavioral boundaries that prevented the agent from entering uncontrolled exploration loops [9].

Dashboard indirection reduces maintenance. In our microservices ecosystem, query structures and metric labels evolve frequently due to new deployments or architectural updates. By dynamically retrieving the current query structure from the dashboard rather than relying on hardcoded queries, the agent eliminated the need for manual prompt maintenance whenever a metric definition changed, thereby enhancing the resilience of the system.

The diagnostic speedup is significant but bounded. Agent Iris reduced the initial diagnostic time from approximately 5-15 minutes to 8 seconds on average, representing an approximate 75-fold improvement. However, the agent's effectiveness is currently limited to domains with documented knowledge in Confluence. Incidents outside the documented scope require human intervention.

5 Conclusion

This paper presented Agent Iris, an intelligent assistant for incident detection deployed at iFood. The empirical evaluation demonstrated that GPT-4.1 provides the optimal balance between response speed (8 seconds on average) and diagnostic accuracy when compared to four other commercial models. The key practical lessons include the importance of grounding agent reasoning in domain-specific documentation written in ubiquitous language, and the operational advantage of embedding tool specifications directly within the agent platform rather than in the knowledge-retrieval source.

The current deployment of Agent Iris serves as a first diagnostic layer, a reactive assistant that accelerates root-cause identification when triggered by a human operator. While no technical blockers are preventing immediate autonomous resolution, the project is an employee-driven initiative executed iteratively alongside daily demands.

The progression from reactive assistant to proactive resolver represents the strategic roadmap for Agent Iris within the iFood operational ecosystem. The long-term vision is to evolve Iris into a proactive agent capable of:

- **Proactive anomaly detection:** continuously monitoring metrics and logs to identify emerging patterns before they escalate into critical incidents;
- **Autonomous resolution:** executing predefined remediation playbooks (e.g., scaling services, restarting consumers) grounded in the RAG knowledge base, without requiring human intervention;
- **Predictive analysis:** leveraging historical incident data and trending metrics to anticipate failures and alert teams preemptively;
- **Quantitative Analysis & Open Dataset:** publishing a characterized dataset of the evaluated incidents (including incident types, complexity levels, and variance analysis) to support a detailed quantitative analysis of the agent's diagnostic accuracy;
- **Performance Evaluation Framework:** establishing a framework to evaluate alternative LLMs and mitigate vendor lock-in continuously;
- **Operational Impact Measurement:** tracking the broader impact of the solution, specifically its effect on Mean Time to Resolution (MTTR) and on-call engineering workload.

Artifact Availability

The artifacts associated with this work, including Agent Iris's source code, prompts, and the dataset of production incidents, are proprietary internal tools and data of iFood and are not publicly available. Although the artifacts themselves are not shared, this paper intends to describe the implementation approach in sufficient detail for software engineers and SREs facing similar challenges to adapt it to their own incident response workflows. The concepts discussed, such as LLM-based observability integration, dashboard indirection, and RAG grounded in ubiquitous language, are not specific to the iFood context and can be replicated using open-source LLM orchestrators and standard observability platforms.

References

[1] Anthropic. 2024. Model Context Protocol (MCP). https://modelcontextprotocol.io.

[2] Atlassian. 2023. Confluence Documentation and Collaboration Platform. https://www.atlassian.com/software/confluence.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, Sebastopol, CA.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[5] Harrison Chase. 2023. LangChain: Building Applications with LLMs Through Composability. https://www.langchain.com.

[6] Eric Evans. 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, Boston, MA.

[7] Gregor Hohpe and Bobby Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Professional, Boston, MA.

[8] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional, Boston, MA.

[9] Yohan Lee, Jisoo Jang, Seoyeon Choi, Sangyeop Kim, and Seungtaek Choi. 2026. Overthinking Loops in Agents: A Structural Risk via MCP Tools. *arXiv preprint arXiv:2602.14798* (2026).

[10] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[11] Sam Newman. 2015. *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media, Sebastopol, CA.

[12] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. ART: Automatic Multi-Step Reasoning and Tool-Use for Large Language Models. *arXiv preprint arXiv:2303.09014* (2023).

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.