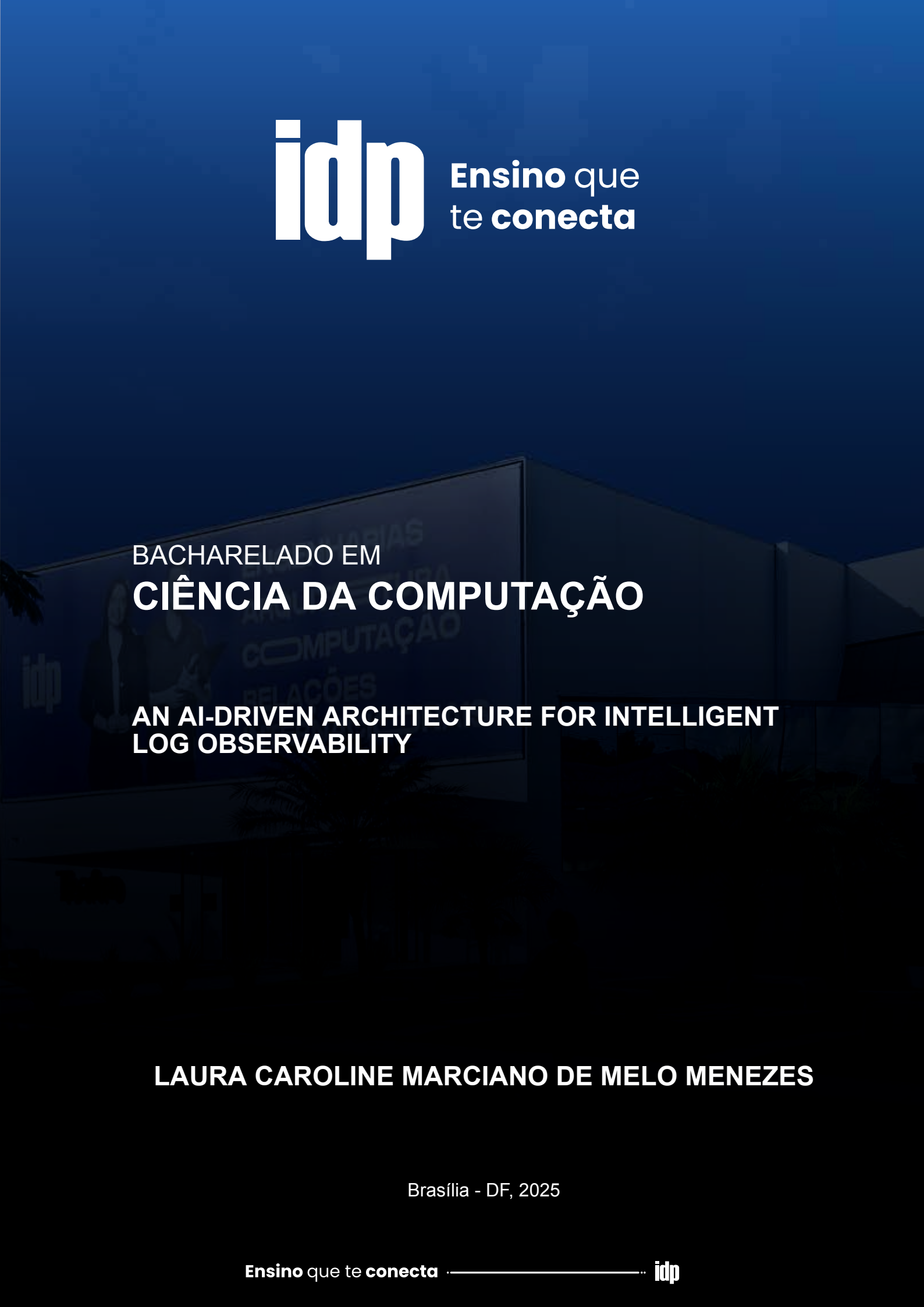


Ensino que te conecta

BACHARELADO EM

CIÊNCIA DA COMPUTAÇÃO

AN AI-DRIVEN ARCHITECTURE FOR INTELLIGENT LOG OBSERVABILITY

LAURA CAROLINE MARCIANO DE MELO MENEZES

Brasília - DF, 2025

LAURA CAROLINE MARCIANO DE MELO MENEZES

AN AI-DRIVEN ARCHITECTURE FOR INTELLIGENT LOG OBSERVABILITY

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção de grau de Bacharel em Ciência da Computação, pelo Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa (IDP).

Orientador
Me. Klayton Rodrigues de Castro

Brasília - DF, 2025

Código de catalogação na publicação – CIP

M543a Menezes, Laura Caroline Marciano de Melo
An AI-Driven Architecture for Intelligent Log Observability / Laura Caroline Marciano de Melo Menezes. — Brasília: Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa, 2025.

68 f. : il. algumas color.

Orientador: Prof. Ms. Klayton Rodrigues de Castro

Monografia (Bacharelado em Ciência da Computação) — Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa – IDP, 2025.

1. Inteligência artificial. 2. Aprendizado computacional. 3. Resolução de problemas - Ciência da Computação. I.Título

CDD 006.31

Elaborada por Natália Bianca Mascarenhas Puricelli – CRB 1/3439

LAURA CAROLINE MARCIANO DE MELO MENEZES

AN AI-DRIVEN ARCHITECTURE FOR INTELLIGENT LOG OBSERVABILITY

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção de grau de Bacharel em Ciência da Computação, pelo Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa (IDP).

Aprovado em 12/12/2025

Banca Examinadora

gov.br Documento assinado digitalmente
KLAYTON RODRIGUES DE CASTRO
Data: 19/12/2025 15:32:15-0300
Verifique em https://validar.iti.gov.br

Me. Klayton Rodrigues de Castro- Orientador

gov.br Documento assinado digitalmente
LUCAS MAURICIO CASTRO E MARTINS
Data: 18/12/2025 16:47:23-0300
Verifique em https://validar.iti.gov.br

Me. Lucas Maurício Martins e Castro- Examinador interno

gov.br Documento assinado digitalmente
LORENA DE SOUZA BEZERRA BORGES
Data: 19/12/2025 14:25:40-0300
Verifique em https://validar.iti.gov.br

Ma. Lorena Bezerra de Souza Borges- Examinadora interna

DEDICATÓRIA

Aos meus pais, Geralda Marciano e Hélio Menezes, pelo amor incondicional, pelos valores transmitidos, por todo o apoio material e emocional, e por serem o meu maior e mais sólido alicerce. Sem eles eu não teria a oportunidade de escrever isso.

Ao meu namorado, Thiago Gough, pela paciência, pela compreensão das minhas ausências, e por ser meu porto seguro e incentivo constante durante toda essa jornada.

AGRADECIMENTOS

A concretização deste trabalho só foi possível graças à colaboração e ao apoio de diversas pessoas e instituições a quem presto, aqui, o meu profundo reconhecimento.

Ao meu Orientador, Prof. Me. Klayton Castro, pela confiança, pela paciência, pela partilha do seu vasto conhecimento e pela orientação crítica e atenciosa que foi fundamental para a excelência e o desenvolvimento desta pesquisa.

Aos professores do curso de Ciência da Computação, que, com dedicação e maestria, transmitiram-me os fundamentos teóricos e práticos essenciais para a minha formação profissional e pessoal.

Aos membros da Banca Examinadora, pela disponibilidade em avaliar este trabalho e pelas sugestões que certamente contribuirão para o seu enriquecimento.

Ao Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa , por proporcionar o ambiente e a estrutura necessários para o meu crescimento acadêmico.

Aos meus amigos e colegas de curso, pela convivência, pelas trocas de ideias e por tornarem a rotina acadêmica mais leve e estimulante.

E, por fim, à minha família, ao meu companheiro e a todos que, direta ou indiretamente, torceram pelo meu sucesso, deixo o meu mais sincero e eterno “obrigado”.

ABSTRACT

Mission-critical distributed applications present significant challenges for diagnosing and interpreting log records due to their high volume, fragmentation, and limited causal structure, especially in environments where failures result from complex interactions rather than isolated events. Metrics-centric monitoring approaches offer limited explanatory support in these contexts, increasing the cognitive load associated with incident configuration, diagnosis, and analysis. This work presents a log-centric observability architecture, designed and evaluated from a real production environment, that integrates structured log ingestion and semantic enrichment. An end-to-end pipeline was implemented based on the Elastic Stack and extended with Small Language Models (SLMs) running locally to enrich raw operational logs with contextual and diagnostic information. Additionally, a numerical prediction experiment was conducted using a traditional supervised machine learning approach, based on runtime signals derived from the logs themselves. The study is based on one year of anonymous production log data collected from a relevant Brazilian public sector system operating under sustained load conditions. The experimental evaluation comprises twelve representative log types, two distinct hardware profiles with CPU-only inference, two compact SLM families, and repeated executions in different configurations, totaling nearly a thousand inference executions. The feasibility of semantic enrichment of records was evaluated considering inference latency, execution stability, and explanatory consistency. The results indicate that semantic enrichment can be applied directly to raw log records with predictable execution behavior on general-purpose hardware, providing a practical basis for log-centric observability architectures in institutional production environments.

Keywords: Observability; Intelligent Operational Diagnosis; Small Language Models; Semantic Log Enrichment; AI for System Operations..

RESUMO

Aplicações distribuídas de missão crítica apresentam desafios significativos para o diagnóstico e a interpretação de registros de log devido ao seu alto volume, fragmentação e estrutura causal limitada, especialmente em ambientes onde as falhas resultam de interações complexas, em vez de eventos isolados. Abordagens de monitoramento centradas em métricas oferecem suporte explicativo limitado nesses contextos, aumentando a carga cognitiva associada à configuração, diagnóstico e análise de incidentes. Este trabalho apresenta uma arquitetura de observabilidade centrada em registros de log, projetada e avaliada a partir de um ambiente de produção real, que integra a ingestão estruturada de logs e o enriquecimento semântico. Um pipeline de ponta a ponta foi implementado com base no Elastic Stack e estendido com Small Language Models (SLMs) executados localmente para enriquecer os registros operacionais brutos com informações contextuais e de diagnóstico. De forma complementar, foi conduzido um experimento de previsão numérica segundo uma abordagem tradicional de aprendizado de máquina supervisionado, com base em sinais de tempo de execução derivados dos próprios registros. O estudo é baseado em um ano de dados de logs de produção anônimos coletados de um relevante sistema do setor público brasileiro, operando sob condições de carga sustentada. A avaliação experimental compreende doze tipos de logs representativos, dois perfis de hardware distintos com inferência apenas por CPU, duas famílias compactas de SLMs e execuções repetidas em diferentes configurações, totalizando quase mil execuções de inferência. A viabilidade do enriquecimento semântico de registros foi avaliada considerando a latência de inferência, a estabilidade de execução e a consistência explicativa. Os resultados indicam que o enriquecimento semântico pode ser aplicado diretamente a registros de log brutos com comportamento de execução previsível em hardware de uso geral, fornecendo uma base prática para arquiteturas de observabilidade centradas em logs em ambientes de produção institucionais.

Palavras-chave: Observabilidade; Diagnóstico Operacional Inteligente; Modelos Pequenos de Linguagem; Enriquecimento Semântico de Logs; IA Aplicada à Operação de Sistemas..

LIST OF TABLES

CONTENTS


1

1 INTRODUCTION

Over the past decade, software systems have evolved toward increasingly distributed, heterogeneous, and data-intensive architectures, resulting in substantial growth in the volume and complexity of operational data and shifting monitoring needs toward the concept of observability, whose definition is the capability to infer a system's internal state from externally produced signals [1, 2].

In contemporary production environments, the continuous generation of large volumes of operational data that must be analyzed to sustain reliability, performance, and availability manifests this trend, where modern applications can emit detailed records of execution behavior, failure conditions, and contextual events across multiple operational signals.

Although indispensable for day-to-day operations, such data becomes increasingly difficult to interpret at scale, particularly in systems that operate continuously and exhibit complex, non-linear interaction patterns, and traditional monitoring approaches, predominantly centered on metrics and static alert rules configured manually by sysadmins and operators, have proven insufficient for diagnosing issues in contemporary distributed systems.

In environments where failures arise from subtle interactions across multiple components rather than isolated faults, threshold-based alerts may indicate degradation but also introduce visual clutter and limited insight into the underlying causes or relationships between events. Additionally, operational logs, although rich in diagnostic content, remain challenging to interpret due to their unstructured nature, fragmentation, and volume. As system complexity grows, manual correlation of log events becomes increasingly error-prone and time-consuming [3].

The expansion of log volume is accompanied by a corresponding increase in operational burden. Large organizations routinely collect and retain massive log datasets to support troubleshooting, auditing, and performance analysis, often relying on platforms such as the Elastic Stack for ingestion, indexing, and visualization [4, 5]. Despite the maturity of these tools, operators are still required to manually inspect events, correlate signals, and reconstruct execution timelines. This manual effort introduces cognitive overload, delays incident resolution, and increases the risk of incomplete or inconsistent diagnoses [1]. These limitations motivate the investigation of mechanisms capable

of augmenting human interpretation rather than replacing it.

Recent advances in Artificial Intelligence (AI), particularly in language-based models, have opened new possibilities for interpreting unstructured operational data. Transformer-based models can process raw textual logs and generate structured summaries or explanations that highlight relevant execution patterns [6–8]. At the same time, the literature consistently reports limitations related to ambiguity, lack of domain grounding, and operational stability, especially in production environments subject to strict reliability, privacy, and resource constraints [9, 10].

These considerations are particularly relevant in on-premises settings, where factors such as data sovereignty requirements, compliance constraints, limited access to specialized hardware, and the need for predictable and fully controlled execution may restrict or delay the adoption of cloud-based inference and external dependencies. In many organizations, these constraints coexist with technical and organizational challenges, including the impracticality of migrating large legacy systems or refactoring complex production environments, rather than indicating an inherent incompatibility with public cloud technologies.

Within this context, this work proposes an observability-oriented architecture focused on log-centric analysis. The architecture combines a structured log-ingestion pipeline with a local semantic-enrichment layer based on compact Small Language Models (SLMs), designed to operate under realistic on-premises constraints. The primary objective is to evaluate whether semantic enrichment applied directly to raw operational logs can support interpretative diagnostic workflows in real production environments. Additionally, the architecture accommodates an auxiliary numerical forecasting feature, applied to signals derived from operational logs and based on traditional machine learning techniques [11].

The proposed design emphasizes architectural decoupling and local execution, explicitly considering the heterogeneous and constrained conditions commonly found in public-sector and enterprise environments. In such contexts, access to specialized hardware is often limited, shared, or reserved for higher-priority workloads, while regulatory, architectural, or data-protection requirements impose constraints on deployment models and execution environments. Consequently, observability solutions must commonly operate predictably on general-purpose infrastructure, coexist with other workloads, and remain deployable within controlled on-premises environments.

The central aim of this study is to design, implement, and evaluate a log-centric observability architecture that integrates semantic enrichment based on compact AI models, enabling efficient and scalable processing characteristics by design, while assessing interpretability, execution stability, and practical viability under on-premises and CPU-only constraints using real production log data.

The specific objectives of this study are:

- To design and implement a log-ingestion pipeline for large-scale collection and indexing of operational logs, with structured enrichment;
- To develop a semantic-enrichment service based on compact AI models, producing interpretable insights from raw operational log events;
- To evaluate the proposed architecture using representative operational scenarios and controlled experiments grounded in real production log data;
- To assess the interpretability, execution stability, and practical viability of semantic-enriched logs under CPU-only execution constraints;
- To conduct a numerical forecasting experiment based on traditional machine learning techniques derived from raw operational log events as an auxiliary analytical component.

The underlying hypothesis of this study is that semantic enrichment performed by compact transformer-based models, when integrated into a structured log-centric observability pipeline, can produce consistent and operationally useful interpretations of raw production logs under CPU-only execution constraints.

This chapter introduced the research context, problem definition, objectives, and hypothesis. The remainder of this work is organized as follows:

- **Chapter 2** presents the theoretical foundations and related work on observability, log-processing challenges, and AI-assisted diagnostic approaches;
- **Chapter 3** describes the methodological approach, including the log-ingestion pipeline, the semantic-enrichment workflow, and the experimental design;
- **Chapter 4** details the proposed observability-oriented architecture and its constituent components;
- **Chapter 5** reports the experimental evaluation and analysis of results, including a complementary predictive modeling scenario;
- **Chapter 6** concludes the study and outlines directions for future work.

2

2 LITERATURE REVIEW

2.1 EVOLUTION OF SOFTWARE AND PLATFORM ENGINEERING

The evolution of modern software architectures has reshaped how systems are designed, deployed, and operated. Before introducing contemporary practices such as DevOps, Site Reliability Engineering (SRE), and observability, it is essential to revisit how traditional operational models functioned and why they became insufficient for increasingly distributed and large-scale environments.

2.1.1 Legacy Monitoring Approaches

In earlier computing environments, system administrators (sysadmins) were primarily responsible for managing complex IT infrastructures. This traditional model involved integrating existing components to form a cohesive service and manually ensuring its stability, including responding to incidents and applying updates [2].

As systems grew in complexity and traffic volume, incidents became more frequent, leading organizations to expand their operations teams. Responsibilities were gradually divided between development and operations due to differing skill sets and priorities [2].

Although this model was widely adopted and supported by a mature ecosystem of tools, it introduced structural challenges. Manual interventions scaled poorly, and the separation between development and operations often caused misaligned goals. Developers focused on rapid feature delivery, while operations emphasized stability. These conflicting priorities led to rigid operational controls, such as detailed checklists and change approvals, and recurring conflicts between teams with distinct points of view [2].

These limitations drove a gradual shift in operational models over the past three decades. Industry practices evolved from reactive, infrastructure-centric monitoring toward integrated reliability engineering and, more recently, telemetry-driven observability [2, 3, 12]. Figure 1 summarizes this progression.

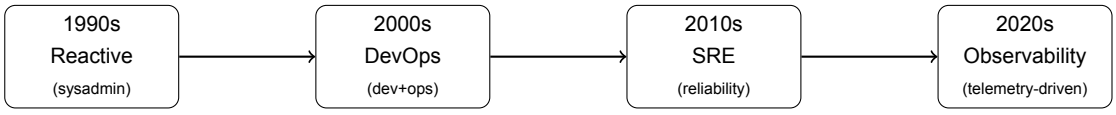


Figure 1: Evolution of Operational Models

This evolution provides the conceptual foundation for understanding why traditional monitoring approaches became insufficient for modern distributed systems and why observability emerged as the next step in platform and reliability engineering [2, 3].

2.1.2 Observability Concepts

To overcome the limitations of traditional monitoring in increasingly distributed environments, modern engineering practices adopted observability-oriented operational models. Rather than focusing on predefined metrics or alerts, observability emphasizes the ability to infer system behaviour from externally observable telemetry, supporting diagnostic reasoning in complex and dynamic application architectures [1].

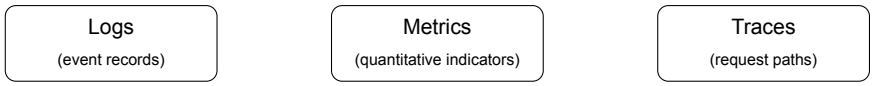


Figure 2: The three pillars of observability

Together, these pillars form the foundation required for diagnosing behaviour in modern distributed systems. However, when treated in isolation—without systematic correlation or contextual interpretation—they reproduce limitations historically associated with traditional Application Performance Monitoring (APM). In contrast, observability-oriented approaches emphasize correlation across heterogeneous data sources and support exploratory analysis in highly dynamic environments [1].

As a critical enabler of platform reliability, observability helps teams identify and resolve problems with reduced manual effort. It enables automated telemetry collection, facilitates debugging, and supports real-time updates to monitoring systems. A highly observable platform can detect and, in some cases, self-correct issues, reducing downtime and improving developer productivity [1].

Observability also underpins DevOps practices by providing data-driven feedback loops for continuous delivery and incident management. It supports root cause detection, feature flag evaluation, and proactive verification using ML techniques. This telemetry feedback strengthens CI/CD pipelines and helps platforms evolve based on real usage patterns [13].

Ultimately, observability enables automated telemetry correlation and contextual analysis to support decisions about system health, reliability, and user experience [1].

2.1.3 Monitoring vs. Observability

To contextualize the conceptual differences between these two operational models, Figure 3 presents a high-level comparison of how monitoring and observability organize their data flows and diagnostic capabilities.

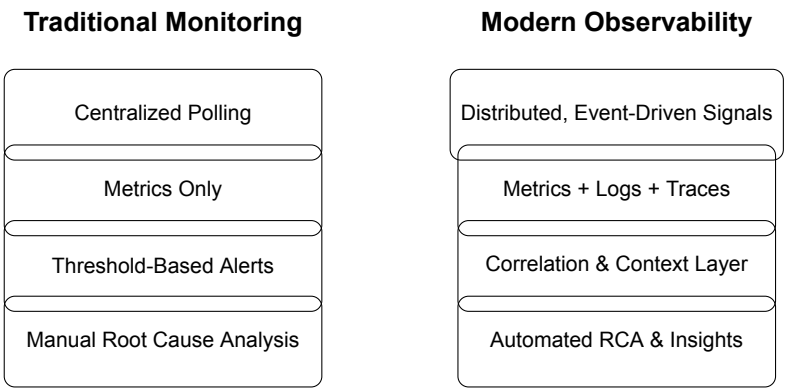


Figure 3: Conceptual architecture comparison between Monitoring and Observability

While Figure 3 summarizes the conceptual structure of both approaches, Table 1 details their practical differences across tools, data types, and troubleshooting strategies.

The table below provides an overview of key differences between traditional monitoring and modern observability. Legacy monitoring practices focus on predefined metrics and system availability through centralized and reactive processes. In contrast, observability emphasizes broader visibility into distributed systems by correlating multiple telemetry sources and supporting more proactive and data-driven operational workflows.

As shown in Table 1, monitoring is reactive and limited to predefined metrics. Observability extends this model by enabling correlated insights across distributed components, supporting more accurate diagnosis and improved operational decision-making.

A wide range of monitoring and observability tools has emerged to support different operational environments. While traditional solutions focus on infrastructure-level metrics, modern platforms integrate telemetry from multiple sources to provide broader diagnostic capabilities. Table 2 summarizes representative tools and their typical usage contexts.

As shown in Table 2, modern observability platforms depend on reliable log-ingestion pipelines to support large-scale telemetry analysis. These pipelines are essential for collecting, transforming, and centralizing operational data, forming the basis for the architecture discussed in the next sections.

Table 1: Comparison Between Traditional Monitoring and Modern Observability Tools

Aspect	Traditional Monitoring	Modern Observability
Focus	Metric collection and system availability	Understanding internal system state from external signals
Typical Tools	Zabbix, Nagios, Cacti, MRTG	Prometheus, Grafana, OpenTelemetry, Elastic Stack
Data Types	Primarily metrics and simple alerts	Metrics, logs, traces, and full-spectrum telemetry
Architecture	Centralized, polling-based	Distributed, agent-based, with event-driven pipelines
Scalability	Limited in high-volume environments	Highly scalable, designed for cloud-native and containerized systems
Troubleshooting Approach	Manual, reactive, rule-based alerts	Automated, proactive, often AI-assisted root cause analysis
DevOps/SRE Integration	Low; infrastructure-centric	High; integrated with CI/CD pipelines and service-level objectives (SLOs)
Problem Resolution Capability	Depends on human expertise	Supports automated diagnosis and prescriptive insights
System Visibility	Component-level metrics (CPU, memory, disk)	Holistic view including services, dependencies, and user experience

Table 2: Comparison Between Monitoring and Observability tools

Tool	Primary Focus	Typical Usage Context
Zabbix	Infra-level monitoring; resource thresholds (CPU, RAM, disk, network)	Legacy systems and static infrastructure with sysadmin-centric operation
Nagios	Health checks, service uptime, plugin-based alerting	Traditional on-premises environments requiring minimal automation
Prometheus	Time-series metrics collection with alert rules; integrates with Grafana	Containerized and Kubernetes-native architectures
Elastic Stack (Elasticsearch, Logstash, Kibana)	Centralized log aggregation, search, and visualization	Hybrid cloud or large-scale systems needing searchable, indexed logs

2.1.4 Log Pipelines and the Elastic Stack

Modern observability practices rely not only on conceptual models but also on robust pipelines capable of ingesting, transforming, and centralizing high-volume operational logs. As applications generate increasingly heterogeneous telemetry—from application logs to infrastructure events—organizations require architectures that reliably capture unstructured data, enrich it with contextual metadata, and make it available for real-time querying and visualization.

Figure 4 summarizes this end-to-end flow, highlighting the main stages through which raw logs are collected, transformed, indexed, and finally visualized.

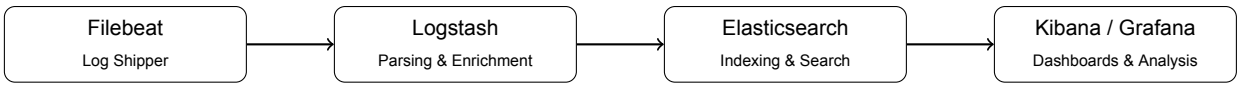


Figure 4: High-level log ingestion and analysis pipeline in the Elastic Stack

As shown in Figure 4, the Elastic Stack organizes log processing into a sequence of specialized components. Each stage plays a distinct role in turning raw, heterogeneous telemetry into structured and searchable operational data.

One of the most widely adopted platforms for large-scale log analysis is the Elastic Stack, a log-centric observability solution composed of specialized components organized as an ingestion and analysis pipeline. Filebeat operates as a lightweight shipper responsible for collecting and forwarding log events with minimal overhead, including

support for multiline reconstruction and secure transport [14]. Logstash provides the transformation and enrichment layer, enabling structural normalization, pattern extraction through Grok and Dissect processors, conditional logic, and metadata augmentation [15].

At the core of the pipeline, Elasticsearch functions as a distributed search and analytics engine optimized for high-volume semi-structured data. By indexing logs as JSON documents backed by inverted indices, Elasticsearch enables low-latency full-text search, aggregations, and correlation across heterogeneous telemetry sources [4]. Visualization and exploratory analysis are supported by tools such as Kibana and Grafana, which integrate directly with Elasticsearch to provide dashboards, contextual inspection, and operational queries [5]. Together, these components transform raw and fragmented logs into a unified, searchable, and analyzable operational data layer.

Despite these capabilities, the Elastic Stack still relies on human interpretation for root-cause analysis, anomaly investigation, and the semantic understanding of complex log patterns. As system complexity and log volume grow, these manual processes become insufficient. This gap between log centralization and log comprehension motivates the integration of Artificial Intelligence into observability pipelines. As explored in the next section, AIOps techniques extend traditional pipelines with automated anomaly detection, semantic pattern extraction, summarization, and contextual reasoning, enabling more intelligent and adaptive operational workflows.

Accordingly, this work does not claim advances in autonomous operations, focusing strictly on interpretability and diagnostic support.

2.2 FOUNDATIONS FOR INTELLIGENT OPERATIONS

As manual log interpretation becomes insufficient in large-scale and highly dynamic environments, the next step is the incorporation of Artificial Intelligence techniques capable of supporting automated reasoning, pattern detection, and adaptive decision-making. This section introduces the main AI concepts required to understand how these capabilities extend traditional observability practices.

2.2.1 Core Concepts of Artificial Intelligence

Artificial Intelligence (AI) is the field concerned with building systems capable of perceiving, reasoning, and acting in pursuit of well-defined goals. It spans subfields ranging from perception and learning to complex tasks such as diagnosis, planning, and natural language understanding. Within this framework, a rational agent is one that selects actions that maximize its expected performance, given its knowledge, goals, and model of the environment [16].

Machine Learning (ML), a central subarea of AI, studies algorithms that improve their performance through experience. ML is especially appropriate when explicit pro-

gramming is impractical—such as in pattern recognition or adaptive system tuning. By learning statistical regularities from data, ML systems can detect patterns, adapt to changing conditions, and make predictions at scale. Over time, the field has incorporated solid theoretical foundations that enable robust performance even in unfamiliar environments [16].

Deep Learning (DL) extends traditional Machine Learning by learning hierarchical representations capable of modeling complex and high-dimensional patterns [16]. Although earlier architectures such as CNNs and RNNs have historical relevance, modern log-analysis and language-understanding workflows rely predominantly on Transformer-based models. This shift reflects their superior ability to capture long-range dependencies, represent semantic structure, and operate effectively in generative or interpretative tasks. Within this context, compact decoder-only Transformers, like the Small Language Models (SLMs), constitute the most practical class of models for CPU-only semantic inference, which is the analytical focus of this study.

A fundamental distinction between traditional programming and Machine Learning lies in how rules are produced. In classical software development, rules are manually specified by the programmer; in ML, rules emerge automatically from data through a training procedure. Figure 5 summarizes this contrast as originally formulated by Mitchell (1997).

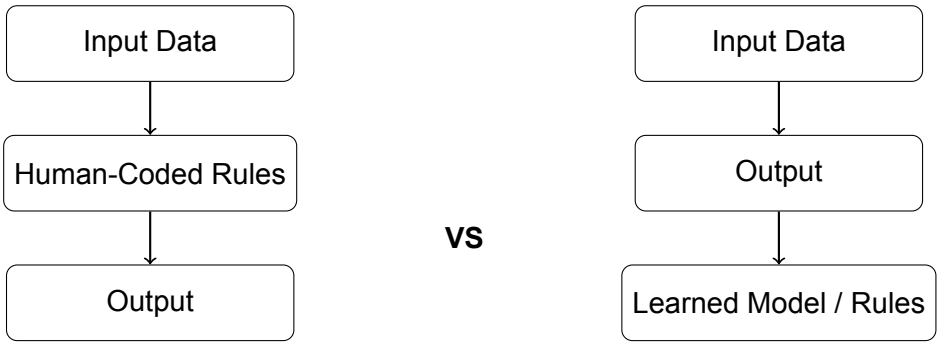


Figure 5: Comparison between rule-based systems and machine-learning models

Machine Learning comprises three fundamental paradigms that define how models acquire knowledge from data. Supervised learning uses labeled examples to map inputs to outputs; unsupervised learning identifies latent structures such as groups or anomalies without labels; and reinforcement learning trains agents through interaction, maximizing cumulative reward under uncertainty [16]. These paradigms provide the conceptual basis for the evolution toward Deep Learning and Transformer-based architectures discussed next.

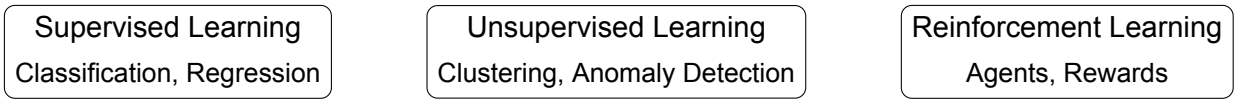


Figure 6: Main Machine Learning Paradigms

The shift to modern generative models begins with the Transformer architecture [6], introduced by Vaswani et al. (2017). Transformers rely solely on self-attention, enabling parallel computation and effective modeling of long-range dependencies. An early influential example is BERT, which uses a Masked Language Model (MLM) objective to recover hidden tokens during training [7]. This encoder-only design remains central to representation learning, classification tasks, and anomaly detection.

In contrast, decoder-only models such as GPT (Generative Pre-trained Transformer) adopt an autoregressive training objective, predicting the next token in a sequence [17]. This generative formulation underpins modern LLMs and their compact variants, the Small Language Models (SLMs), which follow the same architectural principles with reduced computational cost.

More recently, Generative AI (GenAI) has redefined intelligent systems. Built on Transformer-based decoders, LLMs (Large Language Models) such as GPT-4 are trained on massive textual corpora and support general-purpose language understanding and generation [9] [18]. These models typically involve pretraining, task-specific tuning—including Reinforcement Learning with Human Feedback (RLHF)—and deployment for real-time inference. GenAI systems are increasingly applied in domains requiring human-like responses, including IT support, content generation, and diagnostics.

The progression from general AI principles to specialized model families can be summarized hierarchically. Figure 7 illustrates how Machine Learning, Deep Learning, and Transformer-based language models form increasingly specialized subsets within the broader field of Artificial Intelligence.

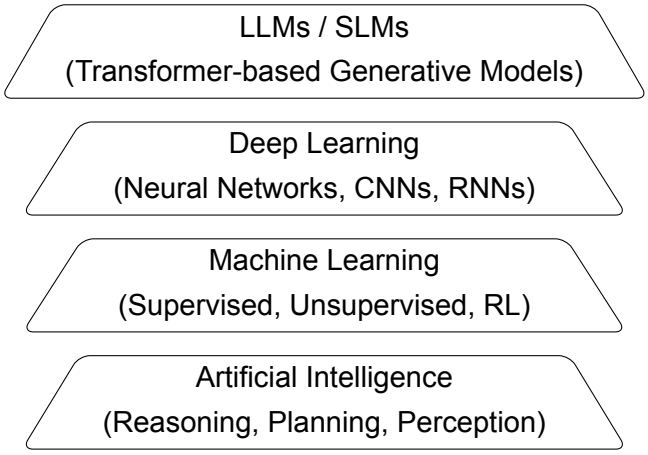


Figure 7: Hierarchy of Modern AI Techniques

Table 3: Comparison between BERT (MLM) and GPT (Autoregressive) Models

Aspect	BERT	GPT
Full Name	Bidirectional Encoder Representations from Transformers	Generative Pre-trained Transformer
Architecture Type	Encoder-only (bidirectional Transformer)	Decoder-only (autoregressive Transformer)
Training Objective	Masked Language Modeling (MLM): predicts masked tokens	Next-Token Prediction (NTP): predicts next token in sequence
Context Direction	Bidirectional: uses left and right context simultaneously	Unidirectional: uses left-to-right context
Main Strength	Representation learning and classification tasks	Text generation and reasoning tasks
Typical Applications	Classification, sentiment analysis, anomaly detection, NER	Dialogue systems, code generation, explanation, summarization
Examples	BERT, RoBERTa, DistilBERT, ALBERT	GPT-2, GPT-3, GPT-4, Phi, Mistral

Within this study, the distinction between encoder-only and decoder-only Transformer models is operationally relevant. Encoder architectures such as BERT excel at producing compact representations for classification or token-level discrimination, but they are not naturally suited for generating coherent explanations from raw log sequences. In contrast, decoder-only models — including compact Small Language Models (SLMs) — operate autoregressively and can synthesize narrative, causal, and context-rich interpretations of operational events. Because this work focuses on semantic interpretability and diagnostic reasoning rather than classification, decoder-only SLMs constitute the appropriate architectural choice, particularly under CPU-only constraints. Table 3 summarizes the main operational distinctions between BERT-like encoder models and GPT-like decoder models.

Together, these AI paradigms provide the theoretical foundation for the techniques adopted in this work. The next section discusses how these capabilities contribute to intelligent operational workflows.

2.2.2 Artificial Intelligence for IT Operations (AIOps)

Artificial Intelligence for IT Operations (AIOps) refers to the integration of AI techniques, such as machine learning, pattern recognition, and automation—into the oper-

ational management of IT systems. This paradigm leverages AI's ability to perceive, reason, learn, and act within complex environments, fulfilling the engineering objective of building intelligent entities capable of decision-making under uncertainty. In modern IT infrastructures—marked by high dimensionality, distributed components, and frequent state changes, AIOps becomes essential for processing telemetry data at scale and supporting diagnostic reasoning beyond human cognitive capacity [16].

The architecture of AIOps is commonly described in terms of three conceptual pillars that mirror the fundamental components of intelligent agents: data ingestion, intelligent analysis, and automation.

- **Data Ingestion:** corresponds to the perception layer in agent-based models. Logs, metrics, and events serve as percepts captured from the environment. Techniques such as parsing, normalization, and information extraction transform raw telemetry into structured representations. As highlighted by Russell and Norvig, perception quality directly determines the quality of downstream reasoning: unreliable or noisy sensor data leads to inaccurate state estimates and suboptimal decisions [16]. In AIOps, this issue manifests as ingestion overload, where massive log volumes require filtering and prioritization to avoid cognitive and computational saturation.

- **Intelligent Analysis:** encompasses the reasoning and learning processes that enable models to interpret operational data. Reasoning derives new conclusions from existing knowledge, while learning improves future performance based on observed patterns. This layer includes anomaly detection, summarization, causal interpretation, and predictive modeling—each contributing to reducing operator effort by structuring or contextualizing raw telemetry.

- **Automation:** reflects the action component of agent architectures, in which analytical outputs may trigger operational responses. In this research, automation is treated solely as a conceptual element. The proposed architecture does not implement autonomous agents, closed-loop remediation, or action-planning mechanisms. Instead, the study focuses exclusively on perception and interpretation, using semantic enrichment to support human-centered diagnostic activities [16].

At the core of intelligent analysis lie the classical paradigms of machine learning—supervised, unsupervised, and reinforcement learning—each contributing differently to pattern identification and anomaly characterization. Supervised learning techniques, such as decision trees, neural networks, and Support Vector Machines (SVMs), are widely used for classifying system states (e.g., normal vs. anomalous) and predicting operational conditions based on labeled historical data. When labeled datasets are scarce, unsupervised learning, particularly clustering, enables the discovery of latent

structures and outliers in unlabeled telemetry. Probabilistic models, including Bayesian networks, support uncertainty management and risk-aware decision-making in diagnostic contexts [16].

Overall, traditional machine learning techniques provide a solid baseline for anomaly detection and failure prediction. Their computational efficiency, interpretability, and stability make them practical for many operational workflows. When logs are properly preprocessed, these models can deliver competitive results and remain important components of intelligent diagnostic systems. In this study, however, their use is intentionally limited to an illustrative forecasting task, while semantic interpretation is conducted exclusively through compact Transformer-based SLMs.

Beyond encoder–decoder distinctions, a broader architectural comparison is required for operational contexts. Under CPU-only constraints, Transformer-based SLMs have gained adoption over recurrent or convolutional architectures due to their superior semantic modeling capacity, parallelizable attention mechanisms, and robustness in handling unstructured textual sequences typical of log data.

The convergence of core AI principles—perception, learning, reasoning, and planning—defines the conceptual trajectory toward intelligent operations. Although the architecture proposed in this study intentionally excludes autonomous action components, the theoretical underpinnings provided by Russell and Norvig frame how future expansions could incorporate transparent decision-making, robust safety mechanisms, and adaptive behaviors aligned with human-defined operational objectives [16].

2.3 RELATED WORKS

Recent advancements in distributed systems and artificial intelligence have driven a shift from passive monitoring to intelligent observability supported by semantic reasoning. This section reviews the most relevant academic and industrial contributions, highlighting how current approaches address log analysis, anomaly detection, and AI-assisted diagnostics. The discussion emphasizes the gap that motivates this work: although the literature explores semantic interpretation using large and compact language models, none provide an open, CPU-viable architecture for local semantic enrichment integrated into a log-ingestion pipeline derived from real production telemetry.

2.3.1 Research Foundations and Related Studies

Bommasani et al. offer a foundational characterization of large-scale foundation models, discussing emergence, transfer learning, and the methodological impact of scaling [9]. While the work is not focused on observability, it establishes the conceptual basis for understanding how language models can generalize across downstream tasks such as summarization, reasoning, and anomaly interpretation — capabilities relevant to semantic log analysis.

Industry analyses from IBM highlight the transition from traditional monitoring toward generative-AI-assisted observability. Their perspective suggests that LLMs can summarize complex event streams, correlate signals, and assist operators during incident investigation [19]. However, these systems remain augmentative rather than autonomous and rely on cloud-based inference, leaving open the question of whether compact, locally executed models can deliver consistent semantic insights without external dependencies — a key objective of this work.

Zhou et al. propose GLog, a self-evolving framework that uses instruction-tuned LLMs combined with clustering to classify anomaly types directly from raw log sequences [20]. Their method avoids traditional parsing and demonstrates that generative models can capture log semantics effectively. Nonetheless, the system requires fine-tuning and clustering pipelines, and does not evaluate inference feasibility under constrained hardware or integrated ingestion environments, as addressed in this research.

Videsjorden et al. develop LUMEN, which integrates multi-agent LLMs with knowledge graphs to support observability in IoT environments [21]. By modeling infrastructure as a semantic graph and dynamically generating analysis code, the system showcases how LLM-driven reasoning can augment operational workflows. However, its reliance on agents, code generation, and large contextual structures differs from this study's focus on compact models performing direct semantic interpretation of logs in CPU-only settings.

Kataria introduces a hierarchical multi-agent LLM framework to automate incident investigation using specialized agents for logs, metrics, and traces [22]. The approach demonstrates that separating reasoning tasks across models improves diagnostic robustness, but it requires cloud-scale computational resources unsuited for local environments. This contrasts with the architectural constraint of the present work, which evaluates lightweight SLMs under realistic on-premise hardware profiles.

2.3.2 Market Solutions and Industrial State of the Art

Commercial observability platforms demonstrate mature but closed—approaches to AI-assisted diagnostics. Dynatrace employs deterministic causal reasoning embedded in its Davis AI engine to compute dependency graphs and perform automated root-cause analysis [23]. While effective, the platform's internal logic is proprietary and unsuitable for experimental evaluation or academic extensibility.

Datadog's Watchdog provides automated anomaly detection across metrics, logs, and traces, leveraging internal ML models and, more recently, generative AI for natural-language investigations [24]. However, as a SaaS solution, it raises concerns regarding data sovereignty, transparency, and the impossibility of local inference evaluation.

Table 4: Comparative overview of related works and market solutions

Work / Platform	Observability / Monitoring	AI / Model Role	Limitations & Research Gaps
Kataria (2025) [22]	Distributed logs, metrics, and traces	Hierarchical multi-agent LLMs	High computational cost; unsuitable for resource-constrained environments.
Zhou et al. (2025) [20]	Log anomaly detection	Self-evolving LLM + clustering	Focused solely on logs; lacks integration with metrics/traces.
Videsjorden et al. (2025) [21]	IoT observability via digital twins	Multi-agent analysis with code generation	Depends on knowledge-graph accuracy; limited scalability in large IoT settings.
Kreuzer et al. (2024) [25]	No observability focus	SLM-as-a-Judge	Compact models reliable in constrained tasks; limited in open-ended reasoning.
Zheng et al. (2023) [26]	No observability focus	LLM-as-a-Judge	Potential evaluator bias; no domain-specific log assessment.
Chan et al. (2023) [27]	No observability focus	LLM-as-a-Judge reliability	Robust in structured tasks; variability across domains.
Bommasani et al. (2022) [9]	No observability focus	Foundation Models	Conceptual scope; no implementation for operational diagnostics.
Dynatrace [23]	Full-stack observability	Deterministic causal AI	Proprietary algorithms; opaque decision logic; high operational cost.
Datadog [24]	Cloud-scale monitoring	Correlation + anomaly detection; generative insights	Closed SaaS; limited transparency; vendor lock-in concerns.
Splunk [28]	SIEM + log aggregation	Predictive analytics and event correlation	High configuration overhead; SPL presents steep learning curve.

Table 4 summarizes the main academic and industrial approaches reviewed, highlighting their observability scope, the role assigned to AI models, and the remaining

research gaps. Although some of the included works are not explicitly focused on observability, they are intentionally incorporated due to their relevance to language-model evaluation, semantic judgment, and methodological validation, which directly underpin the methodological choices adopted in this work.

Splunk remains a dominant platform for Security Information and Event Management (SIEM), a class of systems dedicated to the collection, correlation, and analysis of security-related events for threat detection and compliance [29, 30]. Despite its analytical capabilities, Splunk relies on the proprietary Search Processing Language (SPL) for querying and correlating logs [31]. This dependence introduces a steep learning curve and reinforces vendor lock-in, which, combined with high operational and licensing costs, limits transparency, portability, and extensibility when compared to open and log-centric observability pipelines [32].

Industrial platforms demonstrate advanced AI-assisted observability capabilities, but typically rely on proprietary models, cloud-based processing, or resource-intensive execution environments. As a result, they do not prioritize the evaluation of compact Transformer models executed locally, nor their integration into open and customizable log-ingestion pipelines such as the Elastic Stack, which defines the specific scope and contribution of this work.

Table 4 summarizes the main academic and industrial approaches reviewed, highlighting their scope, analytical focus, and remaining limitations. Although some of the included works are not explicitly focused on observability, they are intentionally incorporated due to their relevance to language-model evaluation, semantic judgment, and methodological validation, which directly underpin the methodological choices adopted in this work.

Industrial platforms therefore demonstrate AI-enhanced observability under tightly controlled conditions, relying on proprietary models whose architectures, training data, and inference mechanisms are not transparent or modifiable by the user. These solutions typically depend on cloud-hosted processing and high-resource execution environments, limiting reproducibility, customization, and on-premises deployment. In contrast, they do not investigate compact Transformer models executed locally, nor their integration into customizable ingestion pipelines such as the Elastic Stack—precisely the gap addressed in this work.

In summary, the reviewed literature shows substantial progress in anomaly detection, semantic log interpretation, and AI-assisted observability. Academic works demonstrate that both LLMs and compact SLMs can extract structure from unprocessed logs, support diagnostic reasoning, and generalize across heterogeneous telemetry. Industrial platforms, in turn, deliver mature solutions but rely on proprietary engines, opaque decision processes, and resource-intensive cloud infrastructures.

Despite these advances, no existing approach provides an open, reproducible, and

CPU-feasible architecture that integrates compact Transformer models directly into a log-ingestion pipeline constructed from real operational telemetry. This gap defines the motivation for the present study: to evaluate small decoder-only models for semantic enrichment within an Elastic Stack workflow, supporting local inference, methodological transparency, and practical applicability under constrained environments.

Taken together, the academic and commercial works reviewed demonstrate significant progress in AI-assisted observability, but they do not converge toward an open and locally deployable solution tailored to constrained operational environments.

2.4 SYNTHESIS AND RESEARCH GAP CONSOLIDATION

The literature reviewed indicates a persistent gap between recent research advances and practical observability deployments. While multiple academic and industrial approaches explore semantic interpretation of operational logs using language models, the integration of lightweight, CPU-efficient models directly into operational log-ingestion pipelines remains insufficiently addressed.

Moreover, existing solutions tend to treat semantic interpretation and numerical telemetry analysis as isolated concerns, without explicitly delimiting their complementary roles within observability workflows. This gap motivates the present study, which investigates the use of compact decoder-only language models for semantic log enrichment within a log-centric pipeline, under realistic on-premises and resource-constrained conditions.

Based on this synthesis, the next chapter presents the methodological design adopted in this work.


3

3 METHODOLOGY

3.1 METHODOLOGICAL DESIGN OVERVIEW

This section provides a high-level overview of the methodological design adopted in this study. Before detailing each methodological component individually, the complete workflow is presented to clarify how data collection, semantic enrichment, and evaluation stages are structurally connected.

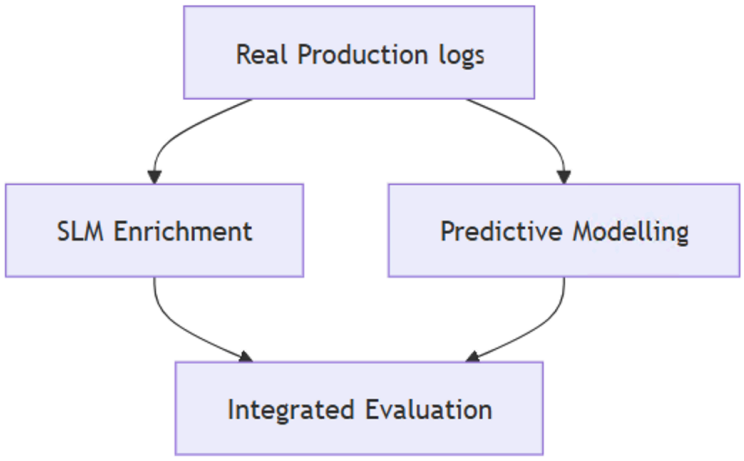


Figure 8: Methodological structure integrating real production logs pipeline with semantic enrichment, predictive modelling and integrated evaluation.

As illustrated in Figure 8, the methodological workflow begins with the collection of one year of operational telemetry from a large-scale government production system, previously anonymized and normalized. The raw log records are then processed by the semantic enrichment pipeline based on Small Language Models, which produces interpretable labels and contextual information not present in the original logs. Within the same architectural workflow, numerical telemetry derived from the operational logs is incorporated as an auxiliary analytical component, where traditional supervised learning techniques are applied to assess the predictability of specific operational phenomena, such as JVM garbage-collection pressure. Finally, the consolidated outputs support the analysis of operational patterns and inform the assisted diagnostic process.

3.2 TYPE AND RESEARCH APPROACH

This research adopts an exploratory, applied, and case-based methodological approach. Exploratory, because heterogeneous and semantically rich operational logs require open-ended analysis to reveal patterns not visible through conventional metrics. It is applied, because the study focuses on implementing and evaluating techniques that can be realistically adopted in IT operations environments. It is case-based because the dataset originates from a complex Brazilian government sector production system, whose behavior cannot be replicated through synthetic workloads. This dataset comprises one year of operational telemetry obtained from a Brazilian public institution and is not publicly available due to confidentiality and institutional restrictions. The data were previously anonymized and normalized through an Elastic Stack pipeline, ensuring ethical handling of sensitive information while preserving the semantic structure required for reproducibility and downstream analysis.

3.3 LOG ENRICHMENT STAGE

Traditional observability pipelines expose numerical indicators such as counters, latencies, and error rates, but they do not explain why events occur or how they relate to broader system behaviour. As distributed environments scale into hundreds of services, anomalies rarely appear in isolation, and operators must manually correlate fragmented, high-volume logs under stringent time constraints. Small Language Models (SLMs) offer a potential mechanism for augmenting this process by producing concise, context-aware explanations directly from raw textual events.

Two compact decoder-only Transformer models were selected for evaluation: Phi-3 Mini and Mistral-7B. This selection reflects a deliberate methodological focus on models that can be executed locally under realistic on-premises and CPU-only constraints, rather than an attempt to exhaustively survey the space of available language models.

Cloud-hosted models, such as GPT-4, were not considered in this study. While these models demonstrate strong generative capabilities, their reliance on external inference infrastructure, proprietary APIs, and opaque execution characteristics introduces constraints related to data locality, reproducibility, and latency predictability that fall outside the scope of the target operational environment [9]. For this reason, the present work concentrates on locally deployable models whose execution characteristics can be directly observed and controlled.

Models from the LLaMA family were likewise not included in the experimental evaluation. Although they support local execution, existing studies report higher memory requirements and increased latency variability under CPU-only inference when compared to more compact architectures, particularly in quantized settings [33, 34]. Given the emphasis of this work on execution stability and predictable behaviour, these models were considered less aligned with the constraints under investigation.

Within this context, Phi-3 Mini and Mistral-7B represent two complementary design points in the space of compact transformer-based models, balancing semantic expressiveness and operational feasibility. Their selection enables a controlled evaluation aligned with the objectives of this study, as summarized below:

- **Deployability**: Both models were executed in 4-bit quantized form, reducing memory footprint and computational cost while preserving sufficient semantic capacity for explanatory tasks. This enables local, CPU-only inference on commodity hardware, consistent with realistic on-premises constraints.
- **Architectural Representativeness**: Mistral-7B and Phi-3 Mini embody distinct design priorities within the class of compact transformer models. Mistral-7B emphasizes broader generative and reasoning capacity, whereas Phi-3 Mini prioritizes efficiency and execution stability. Evaluating both allows comparison across these dimensions without altering the architectural context.
- **Operational Interpretability**: In environments where thousands of heterogeneous log events are generated per minute, the models are evaluated on their ability to condense fragmented evidence into concise, human-readable explanations that support diagnostic reasoning and incident triage.
- **Controlled Evaluation**: Both models were evaluated under identical conditions—using the same log inputs, inference configurations, and CPU-only execution environments—ensuring that observed differences arise from model characteristics rather than experimental artifacts.

This methodological choice is consistent with recent evidence indicating that compact transformer models can deliver stable and interpretable explanations under constrained inference settings, supporting their use in operational observability pipelines [25].

Representative logs were extracted directly from the dataset and supplied to the SLMs without feature engineering, reflecting realistic event and incident-response conditions. Two inference configurations, denoted as A and B, were defined as controlled execution profiles to contrast concise and extended explanatory outputs, enabling the assessment of observable effects on latency and output characteristics under identical execution conditions. The evaluation corpus consisted of twelve representative log instances covering major operational categories, including GC pauses and humongous allocations; SQL timeouts and lock contention; external API timeouts; application-layer exceptions; context-initialization failures; near–Out-of-Memory conditions; and latency anomalies correlated with garbage-collection activity. This selection exposes the models to a broad spectrum of semantic patterns present in the production environment.

Each log instance sampled from the dataset was processed ten times across all combinations of hardware profile, model, and inference configuration, yielding a total of 960 inference executions. For each run, the pipeline collected the following metrics:

- **Latency (ms)**: end-to-end inference time;
- **Tokens-in**: input size derived from the log content;
- **Tokens-out**: length of the generated explanation.

In operational settings, higher input token counts are primarily driven by increased log verbosity, whereas output token counts reflect the verbosity and explanatory depth of the generated responses. Together, these computational metrics complement the qualitative assessment by linking explanatory behaviour to practical deployability within log-centric observability workflows.

3.4 COMPLEMENTARY PREDICTIVE MODELLING

A complementary predictive modelling stage was conducted using Garbage Collection (GC) telemetry. This component serves two methodological aims: (i) to quantify how GC pressure behaves over time in a JVM-based system, and (ii) to establish a baseline illustrating the capabilities and limitations of classical Machine Learning when restricted to numerical telemetry. The GC dataset was reformulated as a binary classification problem: *Will a GC pause occur within a predefined temporal horizon?* Four models were evaluated — Random Forest, XGBoost, LightGBM, and a feed-forward neural network (Multi-Layer Perceptron), using standard supervised-learning evaluation metrics derived from the confusion matrix and ROC analysis [35]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

$$\text{TPR} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{FPR} = \frac{FP}{FP + TN} \tag{6}$$

Accuracy (Eq. 1) measures overall classification correctness. Precision (Eq. 2) and Recall (Eq. 3) capture, respectively, the reliability of positive predictions and the sensitivity to actual positive events. The F1-score (Eq. 4) provides a harmonic balance between Precision and Recall.

All metrics are derived from the confusion matrix, where True Positives (TP) denote correctly identified positive events, True Negatives (TN) represent correctly identified negative events, False Positives (FP) correspond to normal events incorrectly classified as anomalies, and False Negatives (FN) indicate missed positive events.

Receiver Operating Characteristic (ROC) analysis is based on the relationship between the True Positive Rate (TPR, Eq. 5), defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, and the False Positive Rate (FPR, Eq. 6), defined as $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$.

The predictive modeling stage provides quantitative signals derived from the numerical telemetry contained in operational logs. In this context, the evaluated metrics support the analysis of memory-pressure behaviour and help identify periods in which GC pauses tend to intensify, enabling visualization through dashboards and supporting routine operational investigations.

This modeling stage is intentionally limited to numerical telemetry. For free-text log messages, traditional Machine Learning approaches would require additional preprocessing steps such as parsing, vectorization, and feature engineering, including techniques like Term Frequency–Inverse Document Frequency (TF–IDF) and others [36]. In this work, these steps were deliberately avoided in favor of semantic enrichment using Small Language Models, which operate directly on raw textual logs and preserve contextual information without manual feature construction.


4

4 PROPOSED SOLUTION

4.1 ARCHITECTURAL DESCRIPTION

This section presents a structured and technically grounded description of the system architecture. The organization and behaviour of the components are described using a multi-view approach inspired by the arc42 architectural model, specifically the Building Block, Runtime, and Deployment views, as introduced by Starke and Hruschka [37]. This approach is suitable for systems that integrate log ingestion, distributed processing, and AI-based enrichment, as it separates structural, behavioural, and operational concerns, making architectural decisions explicit and verifiable.

4.1.1 Building Block View

The system is organized into functional layers comprising ingestion, storage, analytics, and visualization. Each layer encapsulates a well-defined set of responsibilities. The ingestion layer captures and normalizes operational logs; the storage layer maintains indexed documents and supports low-latency retrieval; the analytics layer performs log-centric semantic enrichment and numerical telemetry analysis; and the visualization layer exposes consolidated information for operational inspection. Figure 9 summarizes these elements and their static relationships.

4.1.2 Runtime View

The Runtime View describes component interactions during execution and the flow of data across the architecture. Logs collected by Filebeat are parsed and normalized by Logstash before being indexed in Elasticsearch. The Inference Engine assumes both producer and consumer roles over Redis: it selects documents from Elasticsearch, enqueues them for processing, retrieves pending tasks, executes local inference using quantized models hosted in Ollama, and stores enriched results back into Elasticsearch. In parallel, numerical telemetry derived from the same log stream is processed by predictive routines operating on structured signals. Figure 10 depicts these interactions.

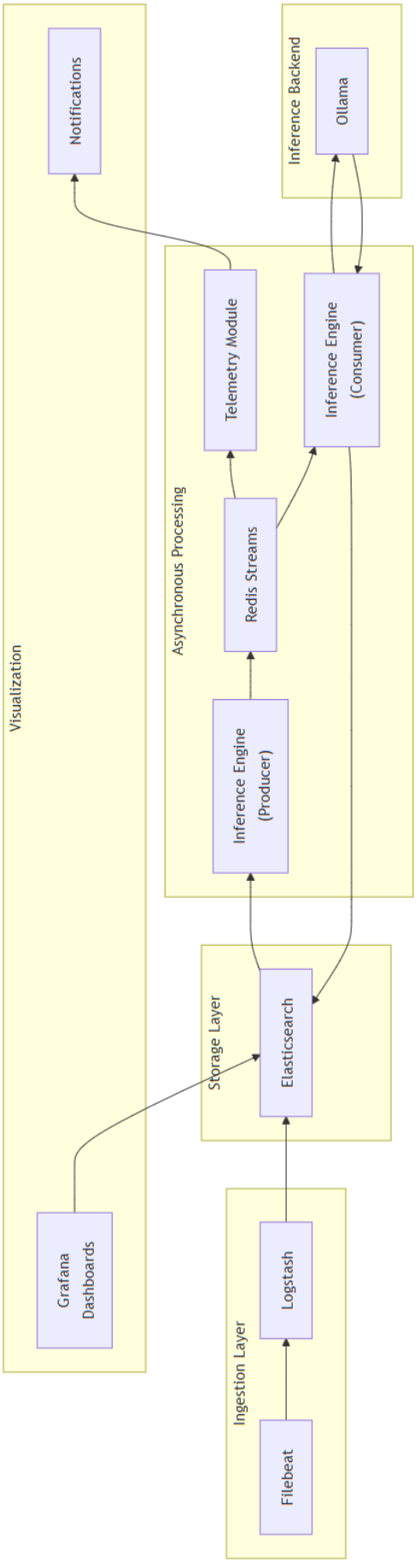


Figure 9: Building Block View of the architecture.

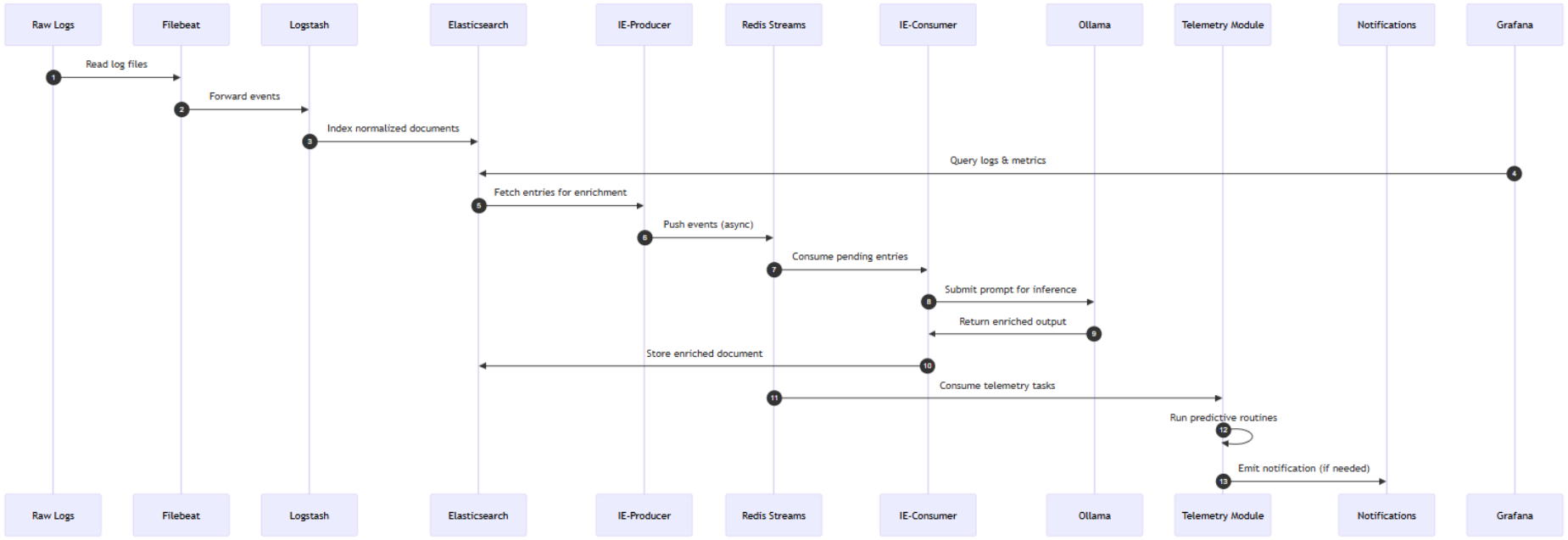


Figure 10: Runtime View showing the operational data flow across components.

4.1.3 Deployment View

All components are deployed as Docker containers on a single host. This deployment strategy reflects typical on-premises virtualized environments in the public sector, where observability workloads operate on shared infrastructure with constrained CPU and memory resources. Consolidating services on a single host reproduces realistic contention patterns and latency sensitivity, enabling the observation of architectural behaviour under CPU-only inference without introducing horizontal scalability assumptions beyond the scope of this work.

Containerization promotes reproducibility and portability by encapsulating each component within a controlled execution environment. Elasticsearch and Ollama rely on persistent volumes to ensure data durability and model availability, while Redis provides centralized asynchronous coordination. Grafana accesses Elasticsearch to support the visualization layer and operational dashboards. The Inference Engine is deployed as independent containers assuming producer and consumer roles over Redis, enabling asynchronous execution without tight coupling between components. Figure 11 summarizes the container-level topology.

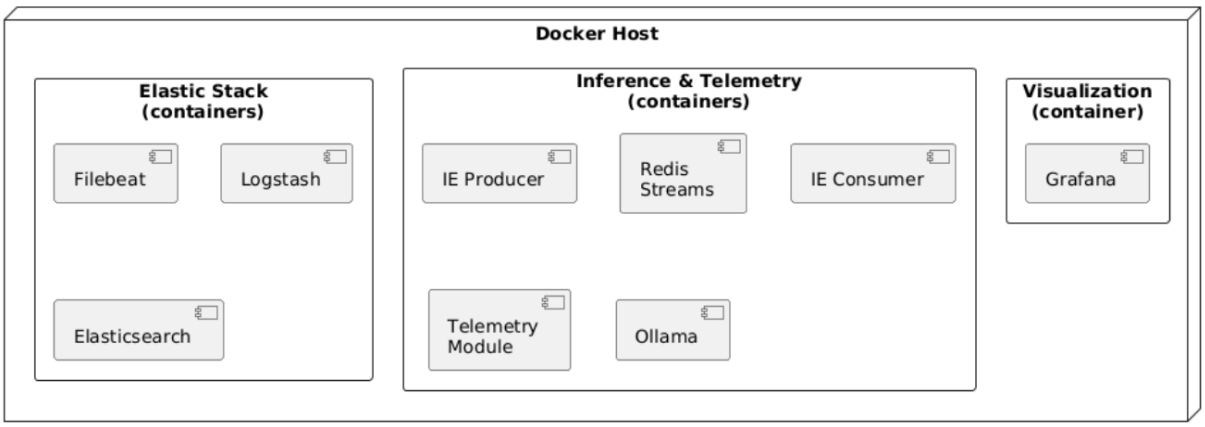


Figure 11: Deployment View of the system as a set of Docker-based services.

4.2 ARCHITECTURAL CONSIDERATIONS AND REPRESENTATIVE SCENARIOS

The architectural design, integration strategy, and operational validation were defined and executed within the scope of this research. All architectural decisions, component integrations, and validation procedures were therefore conducted as part of the research design. The proposed architecture was designed to support log-centric interpretability under constrained on-premises conditions, explicitly accounting for resource contention, asynchronous processing, and analytical consistency. Technology selection follows architectural requirements rather than tool-specific preferences:

- **Elasticsearch** was adopted as the primary storage and indexing component due to its native support for semi-structured log data, time-oriented queries, and low-latency retrieval under sustained write workloads, which are central to log-centric observability architectures.[1]

- **Redis** was employed as an asynchronous coordination mechanism to decouple ingestion, semantic enrichment, and telemetry analysis workloads, providing ordered message processing and backpressure control without introducing heavyweight messaging infrastructures.[2]

- **Inference Engine services**, implemented as lightweight Python–Flask components, act as the execution layer for semantic enrichment, coordinating log retrieval, task dispatch, and CPU-bound inference execution.[3]

- **Ollama** was selected as the local model runtime to enable CPU-only execution of quantized Small Language Models, ensuring predictable inference behaviour, local model control, and compliance with on-premises deployment constraints.[4]

- **Telemetry Module**, also implemented as lightweight Python-Flask components, was included as a dedicated analytical capability responsible for numerical signal extraction and predictive analysis over structured telemetry derived from logs. This module operates independently from semantic enrichment while sharing the same data sources and asynchronous coordination infrastructure.[5]

- **Grafana** was adopted as the visualization layer due to its operational orientation, native integration with Elasticsearch, and interoperability with infrastructure monitoring and alerting ecosystems commonly used in production environments.[6]

Together, these choices support a cohesive and unified architecture aligned with on-premises constraints, asynchronous processing requirements, and predictable operational behavior.

[1] https://www.elastic.co/elasticsearch
[2] https://redis.io/
[3] https://flask.palletsprojects.com/
[4] https://ollama.com
[5] https://scikit-learn.org/stable/
[6] https://grafana.com


5

5 RESULTS AND DISCUSSION

This section presents the execution environment, the dataset of logs, the evaluated models, and the total number of executions. It also summarizes the collected metrics and the objective of the experiment.

5.1 EXPERIMENTAL SETUP

The experiment was executed locally on two different hardware profiles to evaluate the behaviour of small language models under constrained and high-performance CPU environments. All executions were performed using CPU-only inference. GPU acceleration was explicitly disabled through container-level environment configuration, ensuring that all inference was executed exclusively on CPU, following the same operational script and repeating each run ten times per log, per model, and per configuration to measure dispersion and stability.

The dataset consists of twelve operational logs extracted from a Brazilian government production system, previously anonymized and enriched through an Elastic Stack pipeline. These logs represent heterogeneous failure modes, including initialization errors, garbage-collection pressure, timeouts, and mixed degradation scenarios. All logs were submitted to the SLMs using the same prompt structure and the same inference parameters described in the methodology.

Two families of models were evaluated: Mistral-7B (quantized 4-bit) and Phi-3 Mini (3.8B, quantized 4-bit). Each model was tested under two inference configurations (A and B), differing only in generation limits such as maximum tokens and temperature. Both configurations were executed on both machines.

The experiment was conducted on two hardware profiles selected to represent distinct computational conditions relevant to CPU-only inference. The first corresponds to a constrained workstation, while the second reflects a modern high-performance architecture. Their characteristics influence the behaviour of quantized models and therefore must be reported.

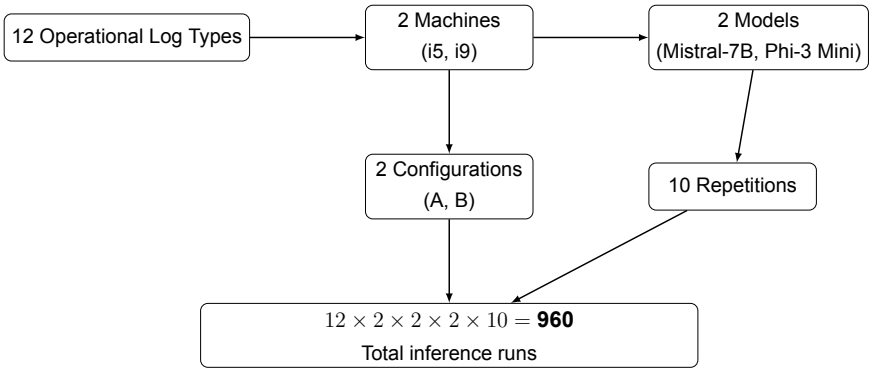


Figure 12: Experimental execution volume across log types, hardware profiles, models, configurations, and repetitions.

The experiment was conducted on two hardware profiles selected to represent distinct computational conditions relevant to CPU-only inference. The first corresponds to a constrained workstation, while the second reflects a modern heterogeneous CPU architecture. Their characteristics influence the behaviour of quantized models and therefore must be reported.

- i5-3470 (Machine 1): Intel processor with 4 cores and 4 threads, base frequency of 3.2 GHz (up to 3.6 GHz turbo), 16 GB RAM, and SATA SSD. This processor supports only the first generation of AVX instructions, which limits vector width and constrains the throughput of low-precision inference kernels.
- i9-14900K (Machine 2): Intel processor with 24 cores (8 performance cores and 16 efficiency cores) and 32 threads, base frequencies of 3.2 GHz (P-cores) and 2.4 GHz (E-cores), boost frequencies up to 6.0 GHz, 64 GB RAM, and NVMe SSD. Performance cores (P-cores) are optimized for high-throughput and low-latency workloads, while efficiency cores (E-cores) are designed for energy-efficient background and parallel tasks. This architecture supports AVX2, enabling wider vector operations and higher throughput for quantized inference.

Clock frequencies are reported as manufacturer base and boost specifications. During execution, dynamic frequency scaling and turbo mechanisms were managed by the operating system scheduler and were not explicitly constrained. These two profiles allow observing how differences in clock frequency, vector instruction support (AVX vs. AVX2), and heterogeneous core design (performance and efficiency cores) affect the behaviour of quantized SLMs, without altering any other component of the experimental setup. All runs were performed using Ollama as the local inference engine. GPU acceleration was explicitly disabled at container level through environment configuration, ensuring CPU-only inference. System monitoring was carried out using standard operating-system tools (`top`, `htop`) and Grafana dashboards to visualize CPU saturation patterns and temporal variability.

Table 5: Summary of experimental setup: hardware, models, configurations, and execution volume.

Component	Specification / Value
Hardware Profiles	i5-3470 (4C/4T, 3.2–3.6 GHz, AVX)
	i9-14900K (24C/32T, P+E cores, up to 6.0 GHz, AVX2)
Models Evaluated	Mistral-7B 4-bit, Phi-3 Mini 3.8B 4-bit
Inference Engine	Ollama (CPU-only)
Configurations	A (short output), B (extended output)
Log Dataset	12 types of production logs, enriched and anonymized
Repetitions	10 runs per log, model, config and machine
Total Executions	960 runs

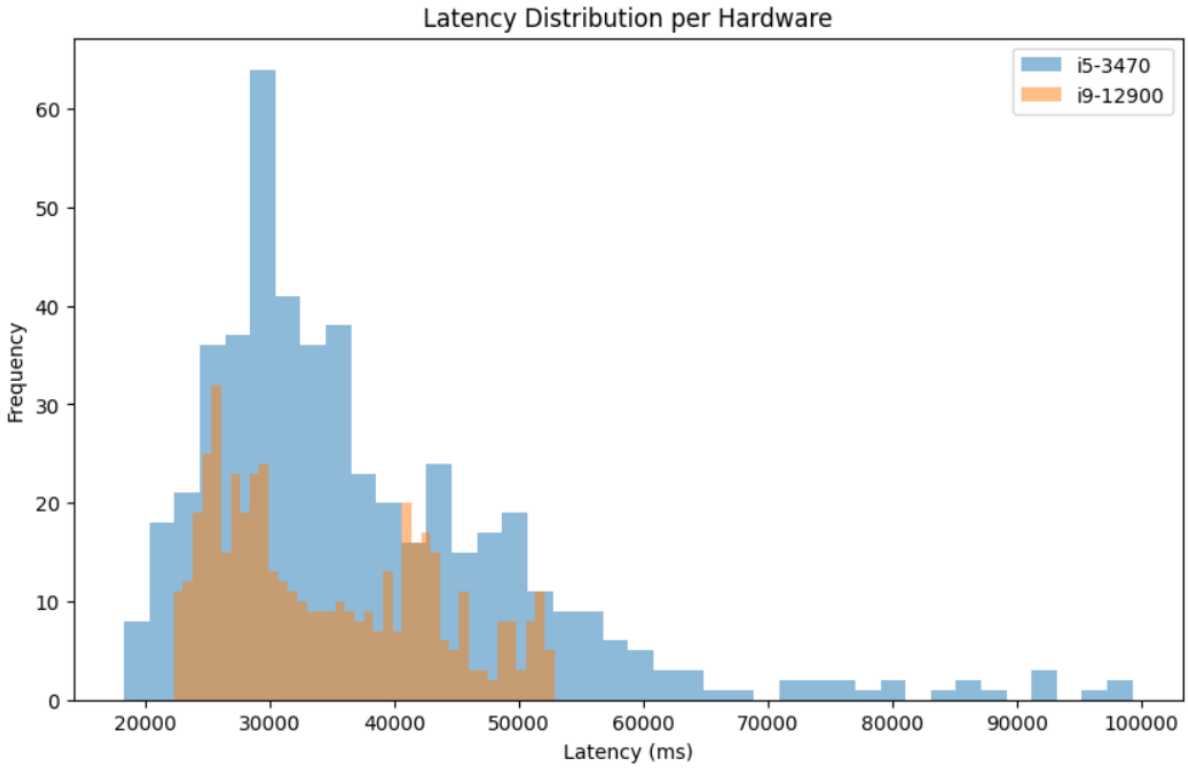


Figure 13: Latency distribution for Mistral-7B and Phi-3 Mini across Machine 1 and Machine 2 hardware profiles.

5.2 INFERENCE PERFORMANCE ANALYSIS

The latency measurements exhibit systematic differences across hardware profiles and model families. Machine 2 consistently achieves lower median latency, reduced dispersion, and absence of long-tail behaviour. By contrast, Machine 1 operates near full saturation, producing higher variance, recurrent spikes, and extended latency tails, especially for Mistral-7B. Phi-3 Mini maintains a compact and predictable latency band

across both machines. Repeated executions confirm negligible variability on the Machine 2 and moderate variability on the Machine 1.

The histogram highlights the contrast between machines: the Machine 2 exhibits a narrow, stable latency band, whereas the Machine 1 shows wide dispersion and long-tail behaviour. Both machines reach 100% CPU usage, but only the Machine 2 briefly recovers between inference cycles. Mistral-7B presents higher medians, wider dispersion, and numerous outliers. Phi-3 Mini remains stable and predictable, confirming that model size is the primary determinant of runtime stability under CPU constraints.

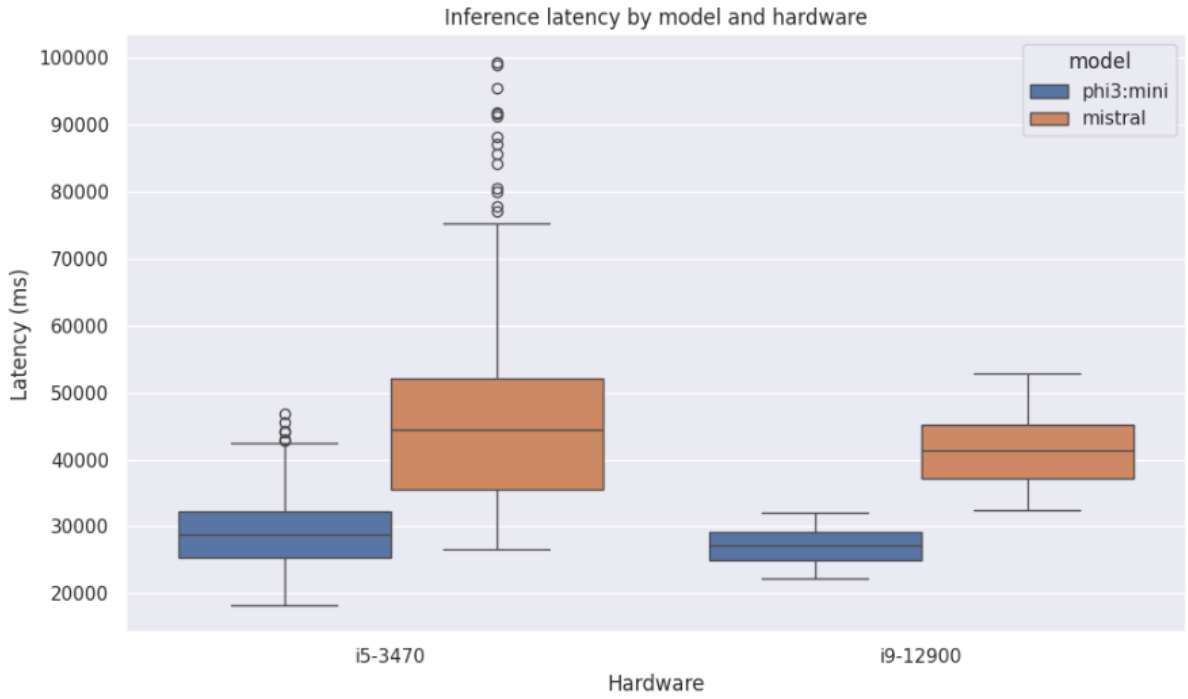


Figure 14: Latency distribution for Mistral-7B and Phi-3 Mini aggregated across all successful executions.

5.3 ASSESSMENT OF EXPLANATORY PERFORMANCE

Before executing the experimental pipeline, exploratory runs were performed to identify decoding parameters that could affect explanation length and determinism under CPU-only inference. These initial tests ensured that the models would operate without stochastic variation that could distort comparative analysis.

Based on these tests, two fixed and deterministic inference configurations were defined. Configuration A limits output depth and sampling flexibility, producing concise explanations with lower computational cost. Configuration B permits longer reasoning chains while maintaining deterministic generation and full comparability across runs.

These configurations act as controlled semantic settings, ensuring that observed differences arise from model architecture and log structure rather than random decoding

behaviour. Because output length was bounded by configuration-specific generation limits, tokens-out remained stable and did not influence latency variability.

The explanatory behaviour of the two models shows clear and consistent differences relevant to operational diagnostics. Mistral-7B generates longer and more elaborated explanations, often integrating events distributed across the log and constructing a broader narrative of the failure scenario. In logs dominated by garbage-collection pressure, this model tends to incorporate multiple log segments and describe memory-related degradation over time.

Phi-3 Mini, in contrast, produces more concise and direct outputs, prioritizing the identification of garbage-collection pressure as the dominant failure mode and its immediate operational implications. This behaviour is well suited for initial diagnostic triage and rapid interpretation under constrained environments.

These distinct explanatory styles are particularly evident in logs containing long temporal sequences, multi-stage cascades, or sustained GC activity. The qualitative differences observed are consistent with the measured variations in output length, latency, and variance reported in the previous section.

Across all executions, both models received identical input prompts and identical log contents, resulting in comparable token-in volumes for each log category. Observed differences therefore arise primarily from output-generation behaviour rather than from variations in input size.

Phi-3 Mini consistently produced shorter and more direct outputs, typically summarizing the dominant failure mode and its immediate operational impact. Mistral-7B generated longer explanations, frequently integrating multiple log segments and describing temporal progression across the execution timeline. This behaviour resulted in higher token-out counts and increased inference latency, particularly for logs dominated by garbage-collection pressure.

This contrast is consistent with expectations reported in the literature on language-model scaling, where larger models tend to generate more elaborated and context-integrative explanations, while smaller models favour concise and directive outputs. In this sense, the observed behaviour of both Phi-3 Mini and Mistral-7B aligns with their respective model capacities and design trade-offs, without exhibiting anomalous or unexpected explanatory patterns.

Both models correctly identified the dominant failure categories present in the dataset, including application initialization faults, memory-pressure indicators associated with garbage-collection activity, repeated GC cycles, timeout manifestations, and external-dependency failures. Rather than providing exact quantitative accuracy measures for semantic correctness, the evaluation focused on qualitative diagnostic adequacy, confirming that the generated explanations consistently captured the primary operational issue and its most relevant consequences, such as response-time degradation or par-

tial service unavailability.

Limitations emerge mainly in logs exhibiting overlapping symptoms or incomplete execution traces. Under these conditions, both models may emphasise secondary effects or simplify complex causal relationships, reflecting the inherent difficulty of interpreting heterogeneous operational logs under CPU-only execution and quantized inference constraints.

Figure 15 shows that inference latency does not scale proportionally with input size. Prompt-encoding cost remains negligible when compared to architectural and hardware factors. Mistral-7B consistently exhibits higher latency than Phi-3 Mini, reflecting its larger parameter count and heavier decoding workload [38]. Machine 2 outperforms Machine 1 across all conditions, reducing both median latency and variance. Larger prompts primarily intensify CPU saturation effects on the constrained hardware profile, without producing a linear growth trend in latency.

Taken together, these results demonstrate that the semantic structure of operational logs and the computational profile of the selected language model jointly determine the feasibility of log-centric semantic enrichment. This empirical evidence directly supports the architectural choices adopted in this study and grounds the conclusions presented in the final section.

Under the evaluated conditions, Phi-3 Mini is operationally viable, while the potential advantages of Mistral-7B can be reassessed in future work involving parameter-efficient fine-tuning and retrieval-augmented generation [39].

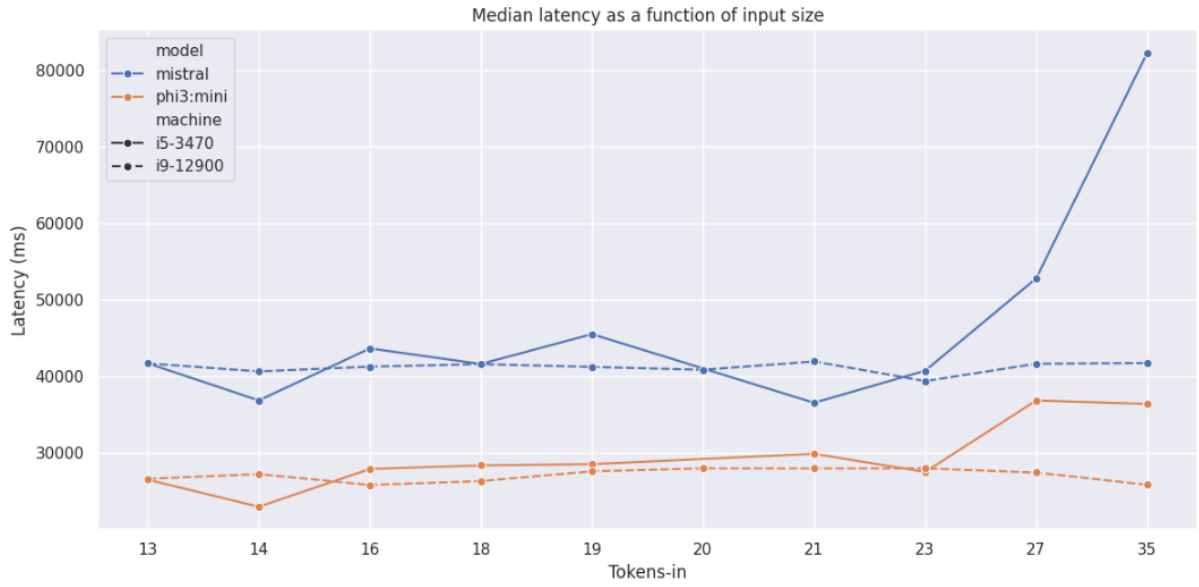


Figure 15: Median latency as a function of input size (tokens-in) across models and hardware profiles.

5.4 MODEL BEHAVIOUR ACROSS LOG CATEGORIES

The twelve logs were grouped according to their dominant operational pattern, based on a technical curation performed by the author. This curation was guided by domain knowledge of Java enterprise applications deployed on application servers such as WildFly, where operational failures in production environments consistently concentrate around two structurally distinct axes: application initialization and JVM runtime execution.

Initialization failures correspond to errors occurring during the bootstrap phase of the application server. In Java EE / Jakarta EE environments, this includes failures related to CDI and bean resolution, dependency injection, datasource configuration, JPA/Hibernate initialization, module loading, and context startup. These failures prevent the application from reaching a stable operational state and typically generate short, dense, and highly deterministic log sequences. Logs dominated by such behaviour were grouped into the Initialization Failures cluster (Cluster A).

Garbage-collection pressure represents the dominant class of runtime degradation in JVM-based systems. Memory exhaustion, near–out-of-memory conditions, repetitive full GC cycles, prolonged stop-the-world pauses, and heap fragmentation emerge progressively during execution and produce long, temporally correlated log sequences. Operational symptoms such as API timeouts, database delays, and thread-pool saturation frequently arise as secondary effects of sustained GC pressure rather than as independent failure mechanisms. Logs dominated by this behaviour were grouped into the Garbage-Collection Pressure cluster (Cluster B).

Based on this architectural understanding, two mutually exclusive semantic clusters were defined: initialization failures and garbage-collection pressure. Each log was assigned to a single cluster according to its prevailing failure mechanism, even when secondary symptoms were present. Timeout-related manifestations were therefore not treated as an independent semantic class, but as effects associated with one of the two primary failure axes. This approach avoids hybrid classifications and provides a consistent analytical structure aligned with the operational realities of Java-based production systems. The resulting classification is summarized in Table 6.

Due to an execution failure during data collection on Machine 1, as shown in Table 16, Log 12 was not fully processed, resulting in a single missing log-hardware combination, where blank cells in the figures indicate unexecuted combinations. All other logs types were executed as planned, and this omission does not affect the overall experimental conclusions. Variability increases when the hardware is saturated, the model is computationally heavy, or the log spans long temporal sequences.

Table 6: Summary of log types, operational characteristics, and semantic clusters.

Log	Type	Synthesis	Semantic Cluster
01	Initialization failure	Bean/Dependency crash	A
02	GC pressure	Near-OOM	B
03	Initialization failure	Context refresh failure	A
04	GC pressure	Repetitive Full GC cycles	B
05	GC pressure	Critical heap usage	B
06	GC pressure	Long GC cycles	B
07	Initialization failure	Bean resolution failure	A
08	Timeout-related failure	API degradation (secondary)	A
09	GC pressure	GC-dominant degradation	B
10	Initialization failure	Bean-related failure	A
11	Initialization failure	Bean-related failure	A
12	GC pressure	Severe memory pressure	B

Initialization-failure logs (Cluster A: 01, 03, 07, 10, 11) are structurally short and contain concentrated error sequences. They consistently produce lower latency for both models, with narrow error bars, particularly on the Machine 2. Their compact structure reduces the explanatory workload of the models, leading to highly stable performance for Phi-3 Mini and only moderate dispersion for Mistral-7B under Machine 1 saturation.

Figure 17 summarises the mean latency and the corresponding run-to-run variability for each log across models and hardware profiles. Beyond architectural differences between SLMs and CPUs, the results show systematic effects associated with the semantic category of each log.

GC-pressure logs (Cluster B: 02, 04, 05, 06, 12) impose a markedly heavier workload. These logs contain extended temporal progressions, repetitive garbage-collection cycles, near-OOM sequences, and long memory-pressure patterns. As a result, both models exhibit higher mean latency and significantly larger standard deviations. Mistral-7B is strongly affected: its latency rises sharply, with wide dispersion on the Machine 1 due to sustained CPU saturation. Phi-3 Mini, although slower than in Cluster A, preserves a compact variance profile, confirming its predictable behaviour in CPU-only inference.

Logs that exhibit timeout symptoms (e.g., API degradation and retry cascades) were not treated as an independent semantic cluster. In JVM-based systems, such symptoms frequently emerge as secondary effects of underlying initialization failures or sustained garbage-collection pressure. Accordingly, logs dominated by timeout manifestations were assigned to the cluster corresponding to their primary failure mechanism, preserving the mutual exclusivity of the classification.

Across all categories, hardware effects remain stable: Machine 2 systematically

reduces inference latency and compresses the error bars, while Machine 1 amplifies variance, particularly for Mistral-7B. Importantly, the relative ordering between log clusters remains consistent across machines and models: GC-pressure logs are always the most computationally demanding; timeout logs show intermediate behaviour; and initialization failures remain the least costly.

These results demonstrate that inference cost is jointly shaped by model size, hardware capabilities, and the semantic structure of the operational log. Logs dominated by long temporal sequences and cumulative state evolution, such as garbage-collection pressure, consistently impose higher computational overhead on SLM-based diagnostic workflows under constrained CPU environments.

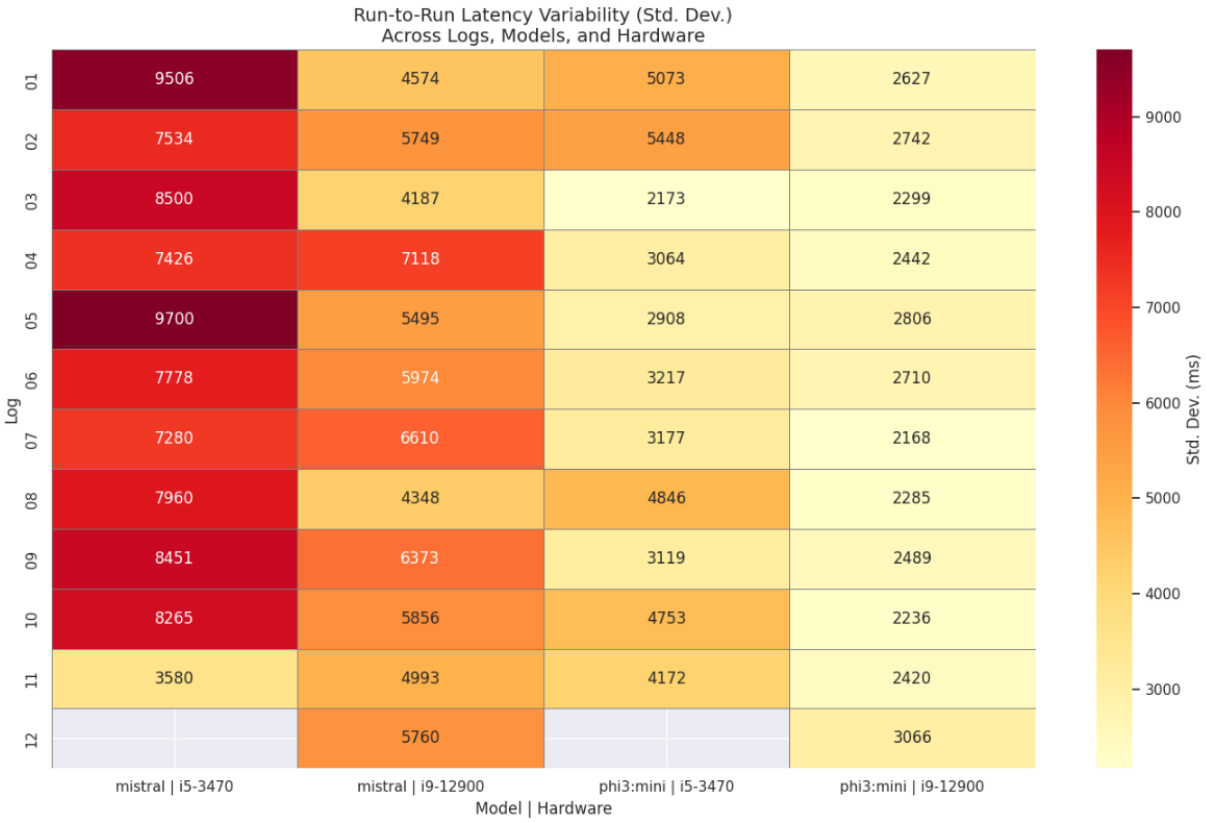


Figure 16: Run-to-run latency variability (standard deviation) across logs, models, and hardware profiles.

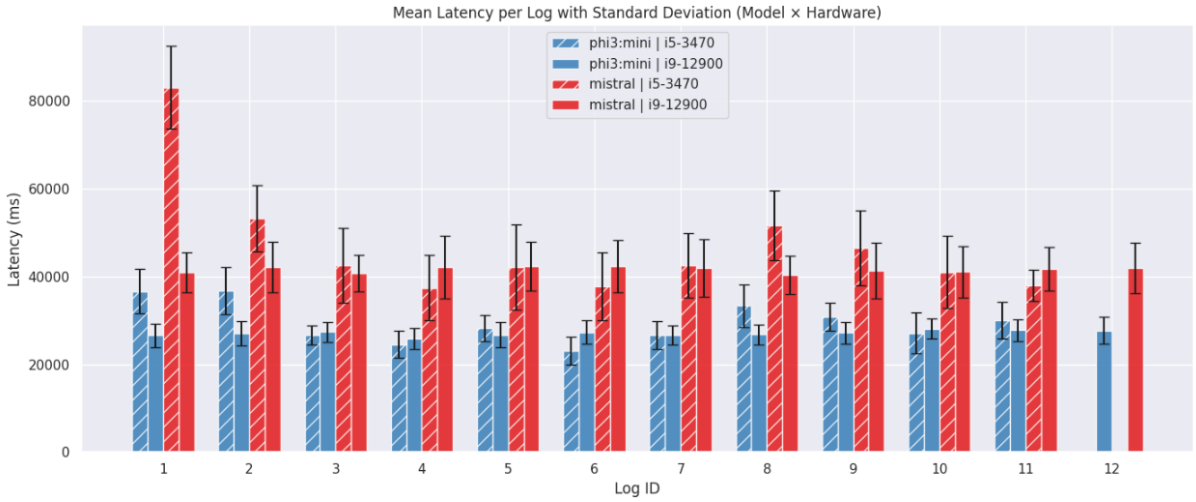


Figure 17: Mean latency with standard deviation across logs, models, and hardware.

5.5 EVALUATION OF THE PREDICTIVE MODEL

Predictive experiments evaluated whether classical ML models can anticipate GC pauses at horizons of 30 seconds, 30 minutes, and 60 minutes, as summarized respectively in Tables 7, 8, and 9. Precision is the critical metric because GC pauses are relatively infrequent events, and each false positive introduces operational noise: it triggers unnecessary alerts, distorts the interpretation of memory pressure, and contaminates dashboards with misleading signals. A model that frequently predicts GC pauses when none will occur imposes real cost on operators by increasing triage effort and reducing trust in monitoring outputs. High precision therefore ensures that positive predictions remain meaningful and actionable, aligning the model's behaviour with the practical demands of observability workflows.

Table 7: Performance metrics for GC forecasting within a 30-second horizon.

Model	Accuracy	Precision	Recall	F1-score
Random Forest	0.87	0.84	0.86	0.85
XGBoost	0.87	0.84	0.85	0.84
LightGBM	0.87	0.83	0.86	0.84
MLP Classifier	0.85	0.82	0.84	0.83

To support a focused and operationally meaningful interpretation of the predictive experiment, two metrics were selected for consolidated visualization: *Precision* and *F1-score*. Precision is the primary metric in this context because GC pauses are relatively infrequent events, and false positives introduce operational noise by generating unnecessary alerts and misleading indications of memory pressure. High precision

therefore reflects the model's ability to issue positive predictions only when they are likely to correspond to actual GC activity, which is crucial for preserving the reliability of triage workflows.

Table 8: Performance metrics for GC forecasting within a 30-minutes horizon.

Model	Accuracy	Precision	Recall	F1-score
Random Forest	0.76	0.92	0.80	0.86
XGBoost	0.66	0.92	0.68	0.78
LightGBM	0.62	0.93	0.64	0.75
MLP Classifier	0.82	0.92	0.88	0.90

Table 9: Performance metrics for GC forecasting within a 60-minute horizon.

Model	Accuracy	Precision	Recall	F1-score
Random Forest	0.88	0.94	0.93	0.93
XGBoost	0.78	0.95	0.81	0.87
LightGBM	0.76	0.94	0.79	0.86
MLP Classifier	0.89	0.94	0.95	0.94

F1-score is included as a complementary metric because it reflects the balance between Precision and Recall, capturing whether the model's behaviour remains stable when both correctness and coverage are considered. While Precision alone indicates the usefulness of alerts, F1-score reveals whether performance gains come at the cost of systematically missing true events. Plotting both metrics together in Figure 18 thus provides a concise representation of each model's operational reliability and overall robustness across the three forecasting horizons.

The patterns observed in Figure 18 highlight how models behave under different temporal horizons. In short windows (30 seconds), Precision and F1-score remain relatively close across all algorithms, reflecting the dominance of immediate memory-pressure effects. At the 30-minute horizon, Precision remains high while F1-score decreases for some models, indicating that positive predictions become more reliable but coverage decreases. In the 60-minute horizon, both metrics converge to higher values for Random Forest and the MLP, evidencing the influence of long-term workload periodicity. Together, these results reinforce that classical ML provides a stable complementary signal to the semantic-enrichment stage, especially when evaluated under metrics aligned to operational relevance.

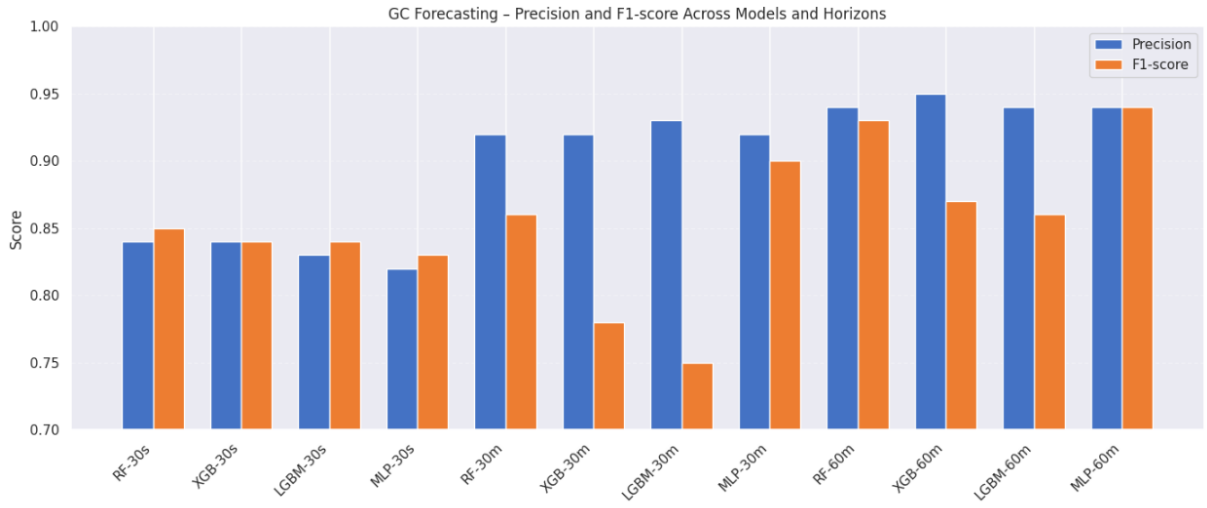


Figure 18: Consolidated comparison of Precision and F1-score across all models and forecasting horizons.

The ROC analysis provides complementary insight into the discriminative capacity of classical models across different forecasting horizons. While short (30 s) and long (60 min) horizons exhibit limited separability, reflected by AUC values close to 0.65, the intermediate 30-minute horizon achieves consistently high AUC values (≈0.90) across all evaluated models. This pattern indicates that GC predictability emerges primarily at temporal scales where memory pressure accumulates and workload periodicity becomes observable. Nevertheless, ROC curves are not treated as the primary decision criterion in this study, since operational relevance in observability contexts depends more strongly on Precision and F1-score than on global discrimination alone.

Figures 19, 20, and 21 illustrate the ROC behaviour across the three forecasting horizons, highlighting the limited discriminative capacity at very short and long horizons and the strong separability observed at the intermediate temporal scale.

The selection of the final predictive configuration follows a hierarchical evaluation strategy grounded in operational relevance. Precision was defined as the primary metric, as false positives in GC prediction introduce unnecessary alerts and degrade trust in monitoring outputs. F1-score was adopted as a complementary metric to ensure that high Precision was not achieved through overly conservative behaviour that would systematically miss true GC events.

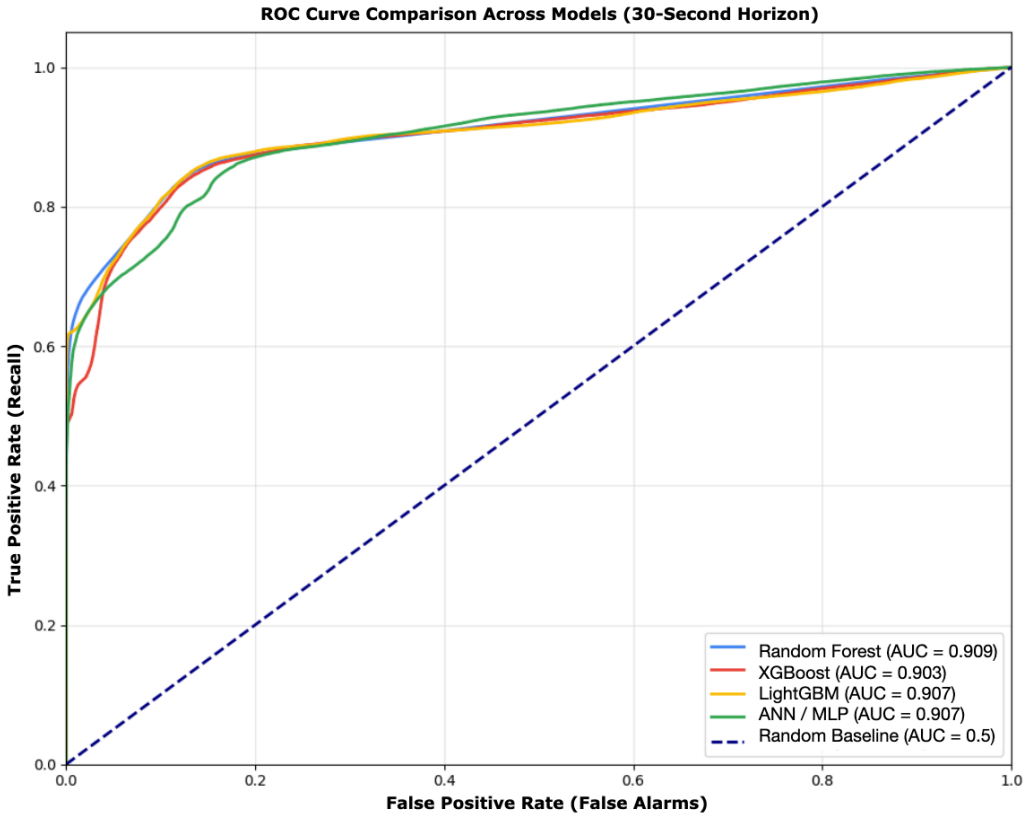


Figure 19: ROC curves for GC prediction within a 30-second horizon.

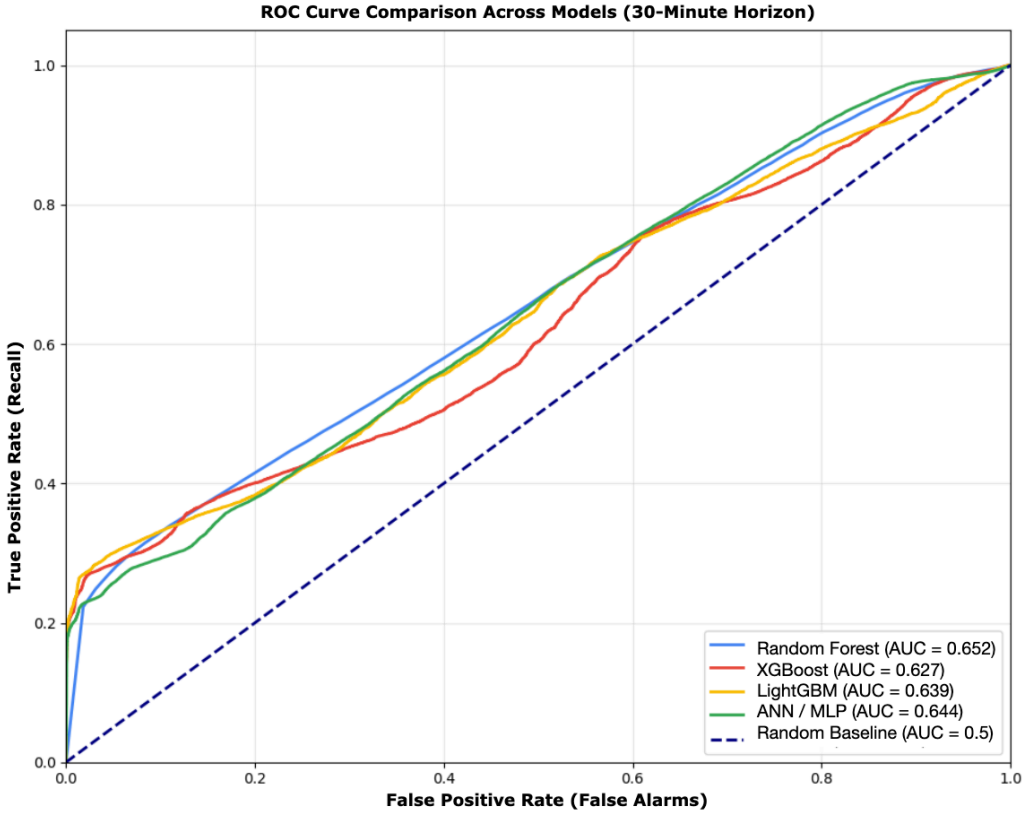


Figure 20: ROC curves for GC prediction within a 30-minute horizon.

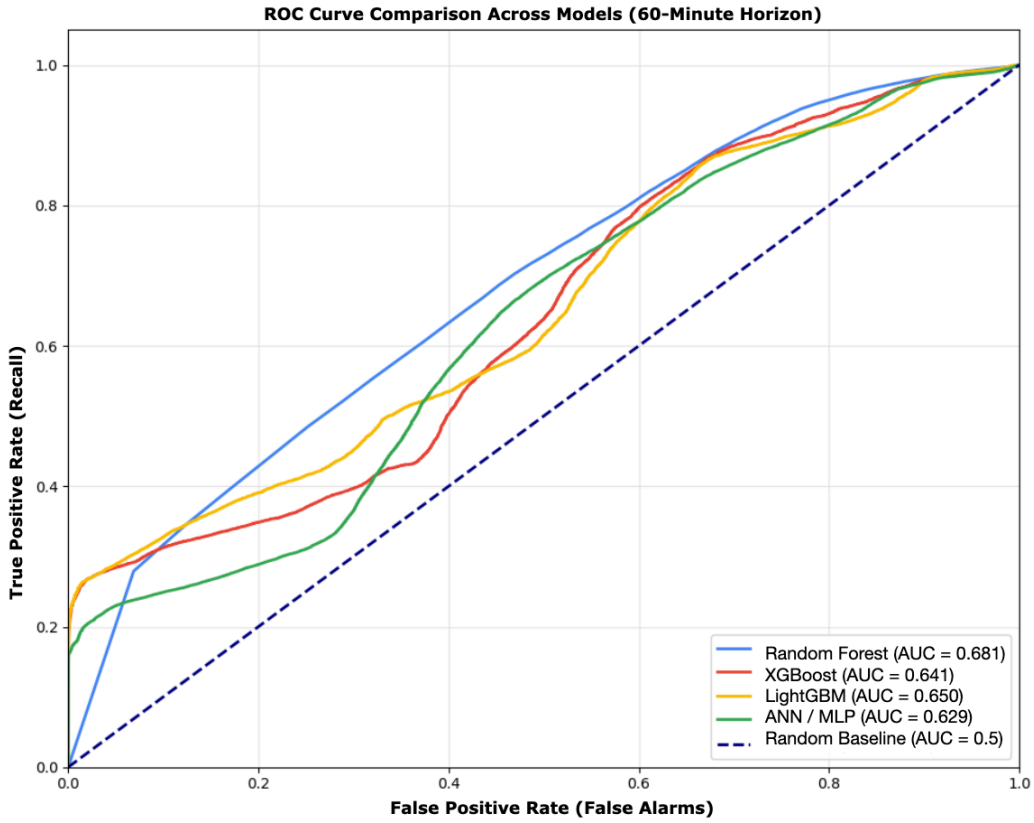


Figure 21: ROC curves for GC prediction within a 60-minute horizon.

Under these criteria, the 30-minute forecasting horizon represents the most suitable temporal scale, as it is the shortest horizon that simultaneously delivers high Precision across all models while providing sufficient lead time for operational response. Within this horizon, the MLP classifier achieved the highest F1-score (0.90) while maintaining consistently high Precision (0.92), indicating a superior balance between alert reliability and event coverage compared to tree-based models, which exhibited reduced Recall and lower F1-scores.

ROC–AUC was used as a final validation criterion to confirm that the selected configuration reflects stable discriminative capacity rather than a threshold-dependent artefact. The high AUC values observed at the 30-minute horizon (≈0.90) confirm that GC predictability is structurally present at this temporal scale. Within this context, the MLP combines strong global discrimination with the most balanced Precision–Recall trade-off. Accordingly, the MLP model at the 30-minute forecasting horizon is adopted as the representative predictive scenario for concluding the experiment.

5.6 OPERATIONAL DASHBOARDS

This subsection presents operational dashboards derived from production logs to provide contextual support for the experimental analysis. The dashboards, implemented in Grafana over data indexed in Elasticsearch, expose time-series behaviour, access

patterns, garbage-collection activity, and application-level errors. They are used to characterise the execution environment and the monitored system, rather than to evaluate model outputs.

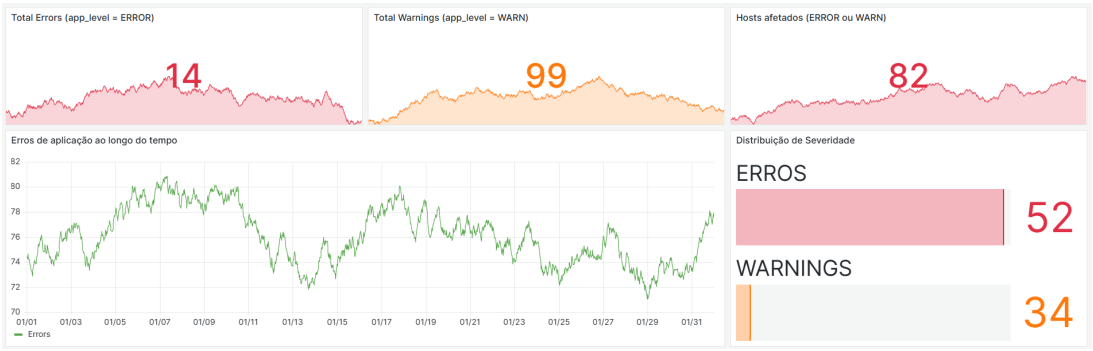
5.6.1 Access and Usage Dashboard Results

Source: Own elaboration.

Figure 22: Access and usage dashboard visualized in Grafana.

The dashboard presents long-term usage patterns, hourly request frequencies, and per-server activity distribution, highlighting natural fluctuations and periods of intensified usage.

5.6.2 Application Errors Dashboard Results



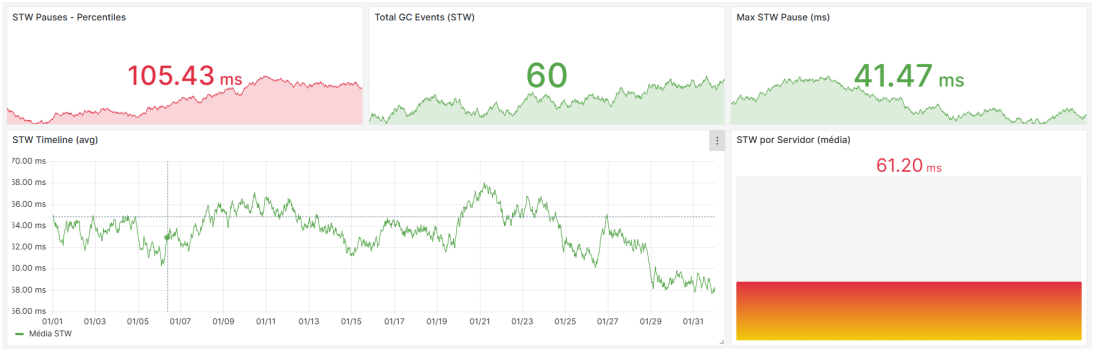
Source: Own elaboration.

Figure 23: Application error and warning dashboard.

The dashboard summarizes total ERROR and WARN events, affected hosts, temporal evolution of errors, and severity distribution.

5.6.3 JVM Stop-the-World (STW) Dashboard

The panels present STW percentiles, event counts, maximum duration, temporal variation, and per-server differences.



Source: Own elaboration.

Figure 24: Dashboard summarizing JVM STW pause behaviour.

5.7 DISCUSSION AND SUMMARY OF FINDINGS

This study examined whether semantic enrichment based on Small Language Models can enhance the interpretability of operational logs collected from a real large-scale production environment under realistic on-premises constraints. The results presented in this chapter indicate that contextual and causal insight can be extracted from raw logs through a log-centric observability architecture, centered on structured ingestion and local semantic enrichment, which constitutes the central contribution of this work and the primary focus of the experimental evaluation.

The evaluated SLMs consistently generated structured explanations that highlighted dominant failure modes, affected subsystems, and plausible propagation patterns. This behavior enables a direct transition from raw log events to interpretative diagnostic understanding, reducing the need for exclusively manual correlation during incident and problem analysis. Under CPU-only execution, Phi-3 Mini exhibited stable latency and predictable execution behavior across hardware profiles, whereas Mistral-7B produced more elaborated explanations at a computational cost that proved less compatible with constrained environments. These observations reinforce model size and CPU characteristics as decisive factors shaping the feasibility of semantic enrichment in on-premises contexts. Resource-monitoring show that inference workloads were predominantly CPU-bound across all evaluated scenarios. CPU saturation consistently preceded any critical memory pressure, while memory usage exhibited transient fluctuations without sustained exhaustion patterns. This behavior supports the architectural emphasis on lightweight, quantized models and CPU-aware execution strategies for shared on-premises infrastructure.

A complementary predictive experiment was conducted as an exploratory analytical step. Using numerical telemetry derived from raw logs, a traditional supervised learning approach was applied to assess whether temporal patterns associated with JVM garbage-collection activity could be identified at different forecasting horizons.

By operating on structured numerical signals rather than textual log representations, this experiment avoids the brittleness of classical NLP pipelines and remains robust to minor log-format variations. The results indicate that traditional machine learning models can provide auxiliary numerical signals in specific scenarios, such as sustained garbage-collection pressure, without competing with semantic enrichment as the primary diagnostic mechanism or requiring heavy pre-processing pipelines based on textual vectorization and retraining.

Taken together, the results presented in this chapter characterize the behaviour of the evaluated log-centric observability architecture under realistic production conditions, establishing an empirical basis for the analysis and conclusions developed in the subsequent chapter.


6

6 CONCLUSION

Operational logs produced by mission-critical distributed systems are fragmented, noisy, and difficult to interpret, particularly in environments where failures arise from complex interactions rather than isolated faults. In such scenarios, metric-centric observability approaches offer limited explanatory power, increasing the cognitive effort required during diagnosis and incident investigation.

This work addressed this problem by designing, implementing, and evaluating a log-centric observability architecture grounded in a real production environment. The proposed solution integrates a structured log-ingestion pipeline with a local semantic-enrichment layer based on compact Small Language Models (SLMs), explicitly designed to operate under on-premises and CPU-only constraints. The study was conducted using one year of anonymized production log data collected from a public-sector system operating under sustained workload conditions.

The experimental results demonstrate that the objectives of this study were achieved. Semantic enrichment based on compact transformer models can be applied directly to raw operational logs with predictable latency, stable execution behavior, and consistent explanatory outputs under CPU-only execution. These findings confirm the practical viability of applying local semantic enrichment based on compact transformer models directly to raw operational logs within log-centric observability pipelines under CPU-only constraints. A numerical forecasting experiment further indicated that quantitative signals derived from logs may provide additional contextual support in specific operational scenarios.

This study is intentionally limited to log-based semantic enrichment and interpretative analysis and does not address automated remediation, autonomous decision-making, or the integration of metrics and traces into a unified reasoning framework. Future work may extend the proposed architecture through parameter-efficient fine-tuning, retrieval-augmented generation, and the incorporation of additional observability signals to expand semantic coverage and diagnostic depth.

Overall, the findings indicate that a decoupled, log-centric architecture centered on local semantic enrichment provides a practical and deployable foundation for interpretative observability, even in complex production environments operating under constrained execution conditions.

REFERENCES

References

[1] IBM, “Observability,” IBM Think Portal, 2024, acesso em: 17 jun. 2025. [Online]. Available: https://www.ibm.com/think/topics/observability

[2] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering*. Sebastopol, CA: O’Reilly Media, Apr. 2016.

[3] S. Niedermaier, F. Koetter, A. Freymann, and S. Wagner, *On Observability and Monitoring of Distributed Systems – An Industry Interview Study*. Springer International Publishing, 2019, p. 36–52. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-33702-5_3

[4] Elastic, “Elasticsearch reference,” Elastic Documentation, 2025, acesso em: 20 jun. 2025. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

[5] ——, “Elastic observability,” Elastic Documentation, 2025, acesso em: 20 jun. 2025. [Online]. Available: https://www.elastic.co/observability

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 6000–6010.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of NAACL-HLT*, 2019.

[8] T. B. Brown, B. Mann, N. Ryder *et al.*, “Language models are few-shot learners,” *arXiv preprint arXiv:2005.14165*, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[9] R. Bommasani *et al.*, “On the opportunities and risks of foundation models,” *Communications of the ACM*, vol. 65, no. 8, pp. 58–67, 2022.

[10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *arXiv preprint arXiv:2005.11401*, 4 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[11] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[12] IBM, “Observability vs. monitoring: What’s the difference?” IBM Think Portal, 2024, acesso em: 17 jun. 2025. [Online]. Available: https://www.ibm.com/think/topics/observability-vs-monitoring

[13] Cloud Native Computing Foundation (CNCF), “CNCF Platforms Whitepaper,” Cloud Native Computing Foundation, Tech. Rep., 2023. [Online]. Available: https://tag-app-delivery.cncf.io/whitepapers/platforms/

[14] Elastic, “Filebeat documentation,” Elastic Documentation, 2025, acesso em: 20 jun. 2025. [Online]. Available: https://www.elastic.co/guide/en/beats/filebeat/current/index.html

[15] ——, “Logstash documentation,” Elastic Documentation, 2025, acesso em: 20 jun. 2025. [Online]. Available: https://www.elastic.co/guide/en/logstash/current/index.html

[16] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence : a modern approach*. Prentice Hall, 2010.

[17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI Technical Report*, 2019.

[18] OpenAI, “Gpt-4 technical report,” OpenAI, Tech. Rep., 2023, arXiv:2303.08774. [Online]. Available: https://arxiv.org/abs/2303.08774

[19] IBM, “How generative ai is changing observability,” IBM Think Insights, 2024, acesso em: 20 jun. 2025. [Online]. Available: https://www.ibm.com/think/insights/observability-gen-ai

[20] J. Zhou, Y. Gao, C. Tan, Y. Yang, and J. Xiang, “Poster: Glog: Self-evolving log anomaly type prediction via instruction-tuned llm and clustering,” in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS ’25)*. New York, NY, USA: Association for Computing Machinery, 10 2025, pp. 4791–4793.

[21] A. N. Videsjorden, H. Song, A. Goknil, D. Roman, and A. Soylu, “Lumen: Enhancing iot system observability with multi-agent large language models and knowledge graphs,” *ACM Transactions on Internet of Things*, 10 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3772077

[22] V. Kataria, “Intelligent site reliability engineering: A multi-agent llm framework for automated incident analysis and root cause determination,” *International Journal of Intelligent Engineering and Systems*, vol. 18, pp. 450–466, 12 2025.

[23] Dynatrace, “Unified observability and security platform,” Dynatrace Website, 2025, acesso em: 07 dez. 2025. [Online]. Available: https://www.dynatrace.com/pt-br/

[24] Datadog, “Cloud scale monitoring and security,” Datadog Website, 2025, acesso em: 07 dez. 2025. [Online]. Available: https://www.datadoghq.com/

[25] D. Kreuzer, A. F. Akyürek, Y. Kim, and J. Andreas, “Small language models can evaluate too: On the feasibility of compact models as reliable judges,” *arXiv preprint arXiv:2402.01719*, 2024.

[26] K. Zheng, W.-L. Chiang, J. Zhuang, M. Wu, C. Lin, H. Zhang, C. Finn, and T. Hashimoto, “Judging llm-as-a-judge: Evaluating large language models as general-purpose evaluators,” *arXiv preprint arXiv:2306.05685*, 2023.

[27] S. W. Chan, S. Santurkar, D. Ganguli, D. Reich, S. Bubeck, J. Gehrke, and P. Liang, “Assessing the reliability of large language models as evaluation judges,” *arXiv preprint arXiv:2310.07298*, 2023.

[28] Splunk, “Unified security and observability platform,” Splunk Website, 2025, acesso em: 07 dez. 2025. [Online]. Available: https://www.splunk.com/

[29] K. Kent and M. Souppaya, “Guide to computer security log management,” National Institute of Standards and Technology (NIST), Tech. Rep. SP 800-92, 2014. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf

[30] Splunk Inc., “What is siem?” 2024. [Online]. Available: https://www.splunk.com/en_us/data-insider/what-is-siem.html

[31] ——, “Search processing language (spl) overview,” 2024. [Online]. Available: https://docs.splunk.com/Documentation/Splunk/latest/Search/AboutSPL

[32] M. Armbrust *et al.*, “The big data systems landscape,” *Communications of the ACM*, vol. 58, no. 12, pp. 38–47, 2015.

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *International Conference on Learning Representations (ICLR)*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[34] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized llms,” *arXiv preprint arXiv:2305.14314*, 2023.

[35] T. Fawcett, “An introduction to roc analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[36] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[37] G. Starke and P. Hruschka, *Software Architecture Documentation in Practice*. Leanpub, 2021. [Online]. Available: https://docs.arc42.org/home/

[38] J. Hoffmann, S. Borgeaud, A. Mensch *et al.*, “Training compute-optimal large language models,” *Nature Machine Intelligence*, vol. 4, pp. 1024–1035, 2022.

[39] P. Lewis, E. Perez, A. Piktus, V. Karpukhin, N. Goyal, and ..., “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *NeurIPS*, 2020.

idp

Ensino que
te **conecta**