IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 February 2025, accepted 17 April 2025, date of publication 29 April 2025, date of current version 8 May 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3565220

RESEARCH ARTICLE

Anomaly Detection and Root Cause Analysis in Cloud-Native Environments Using Large Language Models and Bayesian Networks

DIEGO FRAZATTO PEDROSO[1], LUÍS ALMEIDA[2], LUCAS EDUARDO GULKA PULCINELLI[1], WILLIAM AKIHIRO ALVES AISAWA[1], INÊS DUTRA[2], AND SARITA MAZZINI BRUSCHI[1]

[1]ICMC-Institute of Mathematical and Computer Sciences, University of São Paulo (USP), São Carlos 05508-220, Brazil
[2]Department of Computer Science, FCUP-Faculty of Sciences of the University of Porto, 4169-007 Porto, Portugal

Corresponding author: Diego Frazatto Pedroso (diegopedroso@usp.br)

This work was supported in part by the São Paulo State Research Foundation (FAPESP) under Grant 2019/26702-8, and in part by the Coordination for the Improvement of Higher Education Personnel–CAPES under Grant ROR identifier: 00x0ma614. The authors would also like to thank Mindera for their sponsorship and support throughout this research.

ABSTRACT Cloud computing technologies offer significant advantages in scalability and performance, enabling rapid deployment of applications. The adoption of microservices-oriented architectures has introduced an ecosystem characterized by an increased number of applications, frameworks, abstraction layers, orchestrators, and hypervisors, all operating within distributed systems. This complexity results in the generation of vast quantities of logs from diverse sources, making the analysis of these events an inherently challenging task, particularly in the absence of automation. To address this issue, Machine Learning techniques leveraging Large Language Models (LLMs) offer a promising approach for dynamically identifying patterns within these events. In this study, we propose a novel anomaly detection framework utilizing a microservices architecture deployed on Kubernetes and Istio, enhanced by an LLM model. The model was trained on various error scenarios, with Chaos Mesh employed as an error injection tool to simulate faults of different natures, and Locust used as a load generator to create workload stress conditions. After an anomaly is detected by the LLM model, we employ a dynamic Bayesian network to provide probabilistic inferences about the incident, proving the relationships between components and assessing the degree of impact among them. Additionally, a ChatBot powered by the same LLM model allows users to interact with the AI, ask questions about the detected incident, and gain deeper insights. The experimental results demonstrated the model's effectiveness, reliably identifying all error events across various test scenarios. While it successfully avoided missing any anomalies, it did produce some false positives, which remain within acceptable limits.

INDEX TERMS Automated root cause analysis, Bayesian networks, LLM, cloud computing.

I. INTRODUCTION

Cloud computing concepts and technologies offer significant benefits in terms of scalability and performance. These benefits give developers the ability to rapidly deploy large and diverse types of computing applications.

The name *Cloud Computing* has been used in different contexts over the years. However, it was only in 2006 that Eric Schmidt, then CEO of Google, used the term to describe a new business model that would provide services to users over the Internet [1]. With the continued advancement of technology and the advent of virtualization, companies such as Amazon, Google, and Microsoft established large-scale data distribution facilities, commonly referred to as *data centers*.

Cloud-native applications represent a modern software development paradigm grounded in three fundamental principles: scalability, fault tolerance, and reliability. The rapid

The associate editor coordinating the review of this manuscript and approving it for publication was Chin-Feng Lai.

evolution and proliferation of cloud computing platforms have empowered engineers to design, integrate, and automate sophisticated, large-scale software systems with unprecedented speed and efficiency.

The constant evolution and integration of complex systems technologies have introduced risks associated with performance and reliability, and can compromise the responsibilities assumed by a company or information technology service provider towards customers.

Proper monitoring is one of the most critical components of a high-performance system. Most applications that have their entire life cycle in cloud, from design to discontinuation, are built on microservices architectures. A significant portion of disruptions in a microservices ecosystem can be attributed to inadequate monitoring [2]. Effectively monitoring a microservices-oriented system fundamentally requires three components [3]: (1) the collection of logs and application metrics, (2) dashboards capable of accurately reflecting the current state of the system, and (3) efficient and explicit alerts to identify potential issues within the ecosystem as a whole.

Historically, cloud providers have maintained highly available service levels, often achieving 99.999% uptime. However, it is inevitable that at least one component within an ecosystem will fail at some point. Predicting the timing and location of such failures is a challenging, if not impossible, task. Consequently, availability engineering plays a vital role in ensuring the uninterrupted operation of services, even in the event of component failures.

As the complexity of cloud-native systems continues to grow, the volume of event logs generated by these systems increases exponentially. Consequently, the implementation of a solution leveraging AI becomes essential to detect incident patterns and identify the root cause effectively.Otherwise, developers would be required to manually configure alert rules based on application metrics and logs, which can overwhelm them with false-positive alerts or miss active incidents due to the inherent volatility of these environments.

Previously, developers encountered substantial challenges when manually sifting through extensive log data to pinpoint the root causes of incidents. This labor-intensive process frequently resulted in prolonged system downtime, financial losses, and the inability to uphold the commitments and responsibilities of organizations or service providers.

Historically, logs have been collected and primarily used in a reactive manner to diagnose problems in computer systems. While the data contained within logs is highly valuable, extracting relevant metrics from raw log data has always been a challenge. The continuous evolution of complex systems leads to an exponential increase in the volume of logs generated, particularly as organizations transition towards microservices-oriented cloud architectures. This makes it even more difficult to manipulate, store and extract relevant information from these data sources.

Event-based monitoring is intended to detect unexpected changes in system behavior. However, most existing monitoring solutions do not provide explicit data, leaving developers responsible for identifying outliers in dashboards, defining and adjusting alert rules, and searching through several layers of logs to find the root cause of the issue. Microservices should be treated as living organisms in constant evolution [2] and there are several reasons for this, whether for business or performance reasons. It is common for microservice-oriented systems to have dozens or even hundreds of daily deploys. Constant changes add operational complexity and make it difficult to use predefined thresholds to detect or infer anomalies.

Significant architectural changes to the environment, such as upgrading or removing libraries, performing database migrations, or implementing code modifications, tend to increase the likelihood of outages. All patterns that lead to these events change significantly, making it difficult to use predefined and static thresholds to detect anomalies. Thus, Machine Learning techniques can capture these patterns for any system quickly, and these relationships can scale far beyond the human capacity to keep up with the growing complexity of high-performance systems, helping to decrease the incident time, including: find root cause, mitigation and system repair time.

Nowadays, there are currently dozens of tools for detecting anomalies and root cause incidents. A key objective of these tools is not only to accurately determine the origin of an incident, but also to present the root cause in a clear and concise manner. However, many solutions on the market generate complex and lengthy reports, which can hinder understanding, particularly for users with limited context or expertise across the various layers of the ecosystem and microservices architecture.

This research proposes a dynamic root cause detection system leveraging unsupervised machine learning techniques. The system aims to perform probabilistic inferences using Bayesian networks, adopting a sufficiently generic approach to enable integration into diverse applications while maintaining granularity for use in monitoring systems. The objective is to provide actionable guidance and facilitate automated, precise incident detection.

II. RELATED WORK

We will examine the strengths and limitations of LLMs in this context, comparing their performance with traditional methods and exploring various approaches designed to enhance their effectiveness. The rapid advancement of LLMs [4], [5] has undeniably opened new avenues for anomaly detection, yet significant challenges remain.

This review synthesizes recent research to provide a comprehensive overview of current methodologies, applications, and potential future directions in this exciting and evolving area. The ability of LLMs to process and understand vast amounts of unstructured data, coupled with their capacity for complex pattern recognition and contextual understanding, makes them uniquely suited to tackle the challenges of anomaly detection in a wide range of applications.

However, the inherent limitations of LLMs, such as computational cost, interpretability issues, and potential biases, must be carefully considered and addressed. Log anomaly detection is a critical task for maintaining the security, reliability, and performance of systems across domains such as cloud computing, telecommunications, and cybersecurity. System logs, which record events and behaviors, provide a valuable source of information for diagnosing issues, monitoring performance, and identifying security incidents. Detecting anomalous patterns in logs is vital, as these patterns often indicate system malfunctions, security breaches, or unusual behaviors, enabling timely interventions.

Before delving into LLM-based approaches, it is crucial to establish a baseline understanding of traditional anomaly detection methods and their inherent limitations. These methods have historically relied on a variety of techniques, including statistical methods, rule-based systems, and various machine learning algorithms such as neural networks [6].

While these techniques have proven effective in specific, well-defined contexts, they often struggle when confronted with several key challenges. One significant limitation is their difficulty in handling high-dimensional data, where the complexity of relationships between numerous variables makes it challenging to identify meaningful patterns and anomalies.

Furthermore, many traditional methods require extensive labeled datasets for training, a requirement that can be difficult and expensive to fulfill, particularly in domains where anomalies are rare events.

The reliance on labeled data also limits the generalizability of these models to new, unseen data or different contexts. [7]. Finally, many traditional methods lack interpretability, making it difficult to understand the rationale behind the detection of a specific anomaly. This lack of transparency can hinder trust and acceptance, especially in high-stakes applications such as healthcare or finance. [8]. The emergence of LLMs offers a potential pathway to mitigate these limitations.

Recent advancements in using Large Language Models for log anomaly detection show significant promise. LogFiT, a BERT-based model fine-tuned for recognizing patterns in normal system logs, exemplifies this trend. Unlike traditional methods reliant on predefined templates or supervised training with labeled data, LogFiT uses a self-supervised learning approach.

By predicting masked tokens in log sequences, it captures the linguistic structure of normal logs, making it robust to variations in content and capable of handling out-of-vocabulary tokens. This approach enables LogFiT to identify anomalies by detecting deviations in top-k token prediction accuracy. Experiments on datasets such as HDFS, BGL, and Thunderbird demonstrate that LogFiT outperforms baseline models, particularly in scenarios with high log variability, while integrating seamlessly with the HuggingFace ecosystem for scalable deployment.

Another line of research explores the application of LLMs in time-series anomaly detection. Techniques proposed by Gruver et al. show that LLMs can achieve zero-shot time series forecasting, rivaling or surpassing purpose-built models. By tokenizing numerical data and leveraging LLMs' ability to model multimodal distributions, these methods effectively handle missing data, capture seasonality, and incorporate textual context for predictions. However, challenges such as limited context windows and suboptimal arithmetic performance highlight areas for further research.

In anomaly detection for tabular data, LLMs have been investigated as zero-shot batch-level detectors. Without requiring explicit training on data distributions, these models identify outliers by recognizing low-density regions in datasets. Fine-tuning strategies further enhance the performance of models such as Llama2 and Mistral, with experiments on benchmarks like ODDS showing that fine-tuned LLMs rival state-of-the-art anomaly detection techniques.

Beyond log and tabular data, anomaly detection principles extend to other domains. For example, power systems leverage distributed monitoring and control to maintain stability and observability. Gamboa et al. analyze wide-area monitoring under degraded conditions, emphasizing the criticality of local measurements and the identification of essential system components. This work illustrates the broader applicability of anomaly detection methods to ensure system functionality in diverse contexts.

A. LIMITATIONS OF EXISTING WORK AND CONTRIBUTIONS

Existing works, while effective, often focus on structured or semi-structured data, leaving gaps in the detection of anomalies within unstructured logs. Moreover, most

TABLE 1. Comparison of Recent Works in LLM-Based Log Anomaly Detection.

Approach	Strengths	Weaknesses
LogLLM [9]	Employs a hybrid of BERT and Llama for enhanced semantic understanding; robust to unstructured and natural language logs.	Relies on pre-trained models with high computational demands; pre-processing still requires regular expressions.
LogFiT [10]	Fine-tunes LLMs for structured/semi-structured logs; integrates self-supervised techniques for anomaly detection.	Limited flexibility with unstructured logs; performance heavily depends on labeled data for fine-tuning.
LogBERT [11]	Self-supervised framework for log anomaly detection based on Bidirectional Encoder Representations from Transformers.	Fail to capture semantic information and are unable to handle variability in log content.
MAIA (this work)	Scalable, squema-free approach, low-cost framework for unstructured log anomaly detection using unsupervised LLM fine-tuning.	Potential for high false-positive rates, necessitating refinement to reduce alert fatigue.

approaches struggle with scalability, generalization across heterogeneous log formats, or providing explanations for detected anomalies. This study addresses these gaps by proposing a novel method that extends LLM-based techniques to handle unstructured logs efficiently, incorporating context-aware analysis and scalable deployment strategies.

Table 1 presents a selection of works highlighted in one of the most comprehensive literature reviews published on the topic. It is evident that approaches leveraging large language models (LLMs) are scarce, and none of the reviewed methods utilize LLM insights and Bayesian networks to develop an interactive chat interface for user engagement. The blue lines represent recent studies that also utilize LLMs as anomaly detection engines, but they differ in aspects such as training costs, the structure of training data, and the results achieved.

Log anomaly detection has been a focal point of research, particularly with the advent of large language models (LLMs). Several recent works have contributed novel approaches while highlighting challenges in this domain.

One of the key recent works on anomaly detection using Large Language Models (LLMs) is LogFit. The LogFiT model [10], designed for log anomaly detection, leverages the linguistic capabilities of a pre-trained BERT-based language model, fine-tuned to recognize patterns specific to normal system logs. Unlike traditional log anomaly detection methods that depend on predefined log templates or require labeled data for supervised training, LogFiT adopts a self-supervised approach, learning the linguistic structure of normal log sequences by predicting masked tokens.

LogLLM [9] combines BERT and Llama to enhance semantic understanding of unstructured and natural language logs. Its robust handling of diverse formats is notable, though its reliance on computationally intensive pre-trained models and preprocessing through regular expressions introduces significant overhead [9]. Similarly, LogFiT focuses on fine-tuning LLMs for structured or semi-structured logs and incorporates self-supervised techniques for improved anomaly detection. However, its dependence on labeled datasets limits adaptability for unstructured logs and impedes broader applicability [10].

This makes LogFiT robust to variability in log content and capable of handling out-of-vocabulary tokens. By comparing top-k token prediction accuracy, LogFiT identifies deviations from normal logs sequences and flags them as anomalies. Experimental evaluations on datasets such as HDFS, BGL, and Thunderbird show that LogFiT outperforms baseline models, particularly excelling in scenarios where log content varies. The model's specificity surpasses that of baselines on HDFS and BGL datasets, while maintaining performance on Thunderbird.

Integrated with the HuggingFace ecosystem, LogFiT offers a scalable and adaptable solution for future log anomaly detection tasks, making it a powerful tool for monitoring and maintaining system stability.The preprocessing of logs, such as log parsing, grouping, and representation, can introduce significant overhead [12].

These steps are critical for the effectiveness of the models but may also complicate the deployment process and increase the overall system's complexity. An alternative approach that leverage time series forecasting by encoding numerical time series data as text [13]. The authors demonstrate that LLMs can surprisingly perform zero-shot time series extrapolation at a level comparable to, or even surpassing, purpose-built models.

This success is attributed to LLMs' inherente ability to model multimodal distributions and capture patterns such as seasonality and repetition. To achieve this, the authors propose techniques for tokenizing time series data and converting discrete token distributions into continuous values. Moreover, LLMs can handle missing data without the need for imputation and integrate textual side information to explain their predictions.

While increasing model size generally improves forecasting performance, the paper notes that GPT-4 performs worse than GPT-3 due to tokenization issues and poor uncertainty calibration. Despite limitations such as short context windows and weaknesses in arithmetic tasks, the potential of LLMs in time series forecasting remains promising.

Future research could explore extending context windows, fine-tuning LLMs specifically for time series tasks, and investigating the trade-offs between LLMs' arithmetic limitations and their application to real-world forecasting. The paper positions this work as a step toward unifying various tasks under a single, powerful model, enabling more flexible and scalable forecasting solutions.

A different approach [14] explores the use of LLMs for anomaly detection in tabular data, highlighting their potential as zero-shot, batch-level anomaly detectors. Notably, LLMs can identify outliers by recognizing low-density regions in data without requiring training on a specific distribution. The authors also address the limitations of certain LLMs in anomaly detection by generating synthetic datasets and proposing a robust end-to-end fine-tuning strategy, ultimately improving detection efficacy.

Experiments using the ODDS benchmark[1] show that GPT-4 performs comparably to state-of-the-art anomaly detection methods, even without fine-tuning. Furthermore, fine-tuning models such as Llama2 and Mistral significantly improves their anomaly detection performance. The study highlights the effectiveness of LLMs in identifying anomalies and highlights the value of fine-tuning in enhancing their detection capabilities.

The concept of observability extends beyond cloud computing and applications. Power generating systems must manage numerous variables and rely on models to guide decision-making [15]. analyzes the feasibility of performing wide-area monitoring and control functions in a distributed

[1] https://paperswithcode.com/sota/anomaly-detection-on-odds

way within a power system, leveraging the interconnected system's behavior to describes its dynamics.

Large Language Models represent a significant and promising advancement in the field of anomaly detection. They offer compelling solutions to the limitations of traditional methods, providing enhanced capabilities for identifying complex patterns, handling diverse data types, and generating explanations for detected anomalies. Their flexibility and adaptability make them valuable tools across a wide spectrum of domains, from cybersecurity and finance to healthcare and industrial manufacturing.

In the context of anomaly detection in microservices environments, some mathematical models rely purely on statistical methods, avoiding the complexity of heavy Machine Learning techniques [16]. The wide variety of service types and the diverse nature of metrics add significant complexity to analyzing an entire cluster. Therefore, identifying the most critical metrics is essential for detecting issues and performance bottlenecks in microservices.

While challenges remain in areas such as computational cost, interpretability, and robustness, ongoing research is actively addressing these issues. The continued development and refinement of LLM-based anomaly detection methods promise to significantly transform how we identify and respond to anomalies in diverse applications. This will lead to improvements in efficiency, safety, and security across numerous sectors. Further research into multimodal LLM approaches [17], [18], and the synergistic use of LLMs with other AI techniques [19], will be crucial in unlocking the full potential of LLMs in this rapidly developing field. The development of robust benchmarks and standardized evaluation methods will be essential for facilitating meaningful comparisons between different approaches and driving further innovation.

The proposed MAIA framework seeks to address these gaps by offering an unsupervised, scalable, and cost-efficient solution for unstructured log anomaly detection. By minimizing dependency on labeled data and focusing on fine-tuning LLMs for unstructured logs, MAIA achieves versatility and adaptability. However, like many unsupervised approaches, it requires further refinement to reduce false-positive rates and mitigate alert fatigue.

In summary, while advancements in LLM-based anomaly detection have significantly improved robustness and efficiency, the MAIA framework builds upon these foundations to address key limitations in scalability, adaptability, and cost, making it a strong contender for cloud-native systems.

The literature review helped to gather research sources that will provide a theoretical basis for the continuity of the work. With these analyzes and comparisons, it is possible to perceive which shortcomings and problems still exist in the scope of the work, in order to have a formal proposal for a solution or contribution. The literature review added to the theoretical foundation elements help us to realize that the current tools, in the monitoring context, still have difficulties to generate root cause indicators that are not oblique, that is, any user without context or deep knowledge of the ecosystem as a whole must be able to interpret the root cause of the incident. Based on the review and on all the theoretical foundations, the next chapter presents the proposal that fills the gap left by these works.

III. SETUP

Cloud-native environments are composed mostly of managed elements by service providers. They tend to be fragmented, distributed and have several layers of abstraction, making it easy for the end user [2]. Given this context, we must simulate behaviors and environments compatible with these scenarios. We created a service management and test creation platform based on the Amazon Web Services (AWS) service provider to reach the goal. This environment has previously been utilized in other relevant experiments, maintaining the same context of dynamic event analysis within cloud platforms [20].

Next, we will explain in further detail how this environment works and what components make up the complete system, shown in Figure 1.

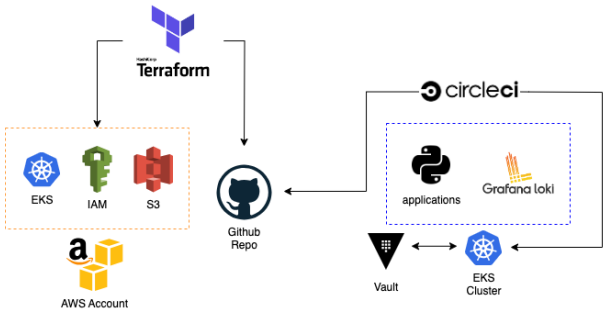

FIGURE 1. AWS Platform-as-a-Service Top-Down.

In the PaaS context, the challenge is to ensure some assumptions: allowing large-scale experimentation by users, guaranteeing consistency of executions, controlling operations, controlling costs, and guaranteeing a governance model [21].

To increase control, we need to have well-structured processes for creating resources for low-level infrastructure, such as AWS components such as EC2, S3, etc., and for application deployments. Next, we will detail how the creation and changes of resources of these two layers work and how they are managed to guarantee all these previously mentioned criteria.

IV. MAIA ARCHITECTURE

To simulate a real-world distributed system, we used the sock-shop application developed by Istio.[2] This application is widely used in cloud-native environments as a reference architecture for microservices, replicating the functionality of an e-commerce platform.

[2] https://github.com/istio/istio

It is often used conjunction with Istio, a leading service mesh framework, to demonstrate key features related to the management, security, and observability of distributed microservices architectures.

The sock shop stack uses the following languages and tools:

- **Programming Languages**: Java, JavaScript and Go.
- **Frameworks**: Spring Boot, NodeJS, and Go kit.
- **Persistence**: MySQL, MongoDB, and RabbitMQ.

Thus, we were able to represent a real production system with several programming languages, frameworks, components, and technologies that will be dynamically analyzed in a distributed environment. All these sub-systems are integrated into the sock shop application as a whole. As shown in Figure 2, the original design featured an ecosystem with several microservices written in different different languages. In the version used for our experiments, the applications in C and. NET are no longer available.

FIGURE 2. Socks-Shop Polyglot architecture[3]

Figure 2 illustrates our experimental system, which depicts a virtual store utilizing a diverse stack of programming languages, frameworks, databases, and more. This system is integrated with the platform developed in a recent paper [16]. By utilizing the PaaS model provided by our platform, users can seamlessly modify and conduct experiments in a secure, parallel, and controlled environment. This approach promotes agility, fosters innovation, and enables risk-free testing, ultimately enhancing the development and optimization processes.

Amazon EKS orchestrates these workloads within the AWS public cloud. To collect logs in their raw format—without structure or parsing—we used the open-source tool Loki[4] from Grafana Labs.

Loki is a horizontally-scalable, highly available, multi-tenant log aggregation system inspired by Prometheus. Unlike Prometheus, which focuses on metrics, Loki concentrates on logs and collects them via a push model rather than a pull model. It was also designed to be cost-effective and highly scalable. Unlike other logging systems, Loki does not index the contents of the logs; instead, it indexes only metadata associated with the logs, using a set of labels for each log stream.

A log stream is a set of logs that share the same labels. Labels are essential for helping Loki locate a log stream within the data store, making the quality of labels crucial for efficient query execution.

For long-term data retention at a lower cost, Amazon S3 was utilized as the storage backend, as S3 storage costs are significantly lower than those of EC2-attached disk volumes.

A. LOAD AND CHAOS TEST

Locust[5] was used as the open-source load testing tool to evaluate the performance and scalability of web applications and other services. It facilitated the simulation of numerous concurrent users interacting with the system, enabling an assessment of its performance under varying load conditions. The results generated from load tests conducted with Locust were seamlessly integrated with Grafana, providing real-time visualization and analysis of the outcomes.

In addition to generating synthetic user load, it is essential to simulate a wide range of potential issues that the infrastructure or application might face in real-world scenarios.

Chaos Mesh[6] was the open-source chaos engineering platform used in our Kubernetes environments. It enables developers and operators to inject a variety of faults into their applications, enabling the assessment of resilience and the capacity to endure unexpected conditions.

In Figure 3, the interconnections between the components are illustrated, highlighting their relationships and the overall workflow.

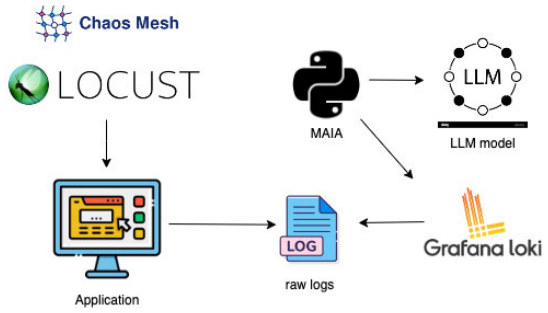

FIGURE 3. MAIA Architecture.

V. SETUP

The sock-shop application consist of 12 microservices in total, each serving distinct purposes and functioning together as part of the overall system - the socks store. All microservices are listed in 2.

[3]https://redthunder.blog/2018/07/30/socks-shop-polyglot-app-in-kubernetes

[4]https://grafana.com/oss/loki/

[5]https://locust.io/

[6]https://chaos-mesh.org/

TABLE 2. Microservices in the Sock-Shop Application.

Microservice	Description	Implementation
Payment	Processes payments for orders.	Go
Queue-master	Manages message queues.	Java
Catalogue	Lists available products.	Go
Catalogue-db	Stores catalogue data.	MySQL
RabbitMQ	Facilitates asynchronous messaging.	Java
User	Manages user authentication.	Node.js
User-db	Stores user authentication.	MongoDB
Carts-db	Stores cart data.	MongoDB
Carts	Manages shopping cart operations.	Java
Front-end	User interface for interaction.	Node.js
Orders-db	Stores order information.	MongoDB
Orders	Manages order creation.	.NET / Java
Shipping	Handles shipping logistics.	Java
Session-db	Stores user session data.	Redis

Load tests lasting approximately one hour were conducted to train the model. During these tests, only the introductory load generated by Locust was applied, ensuring that no anomalies or errors were reported by any application. The load test followed a standard workflow: a user was created, random products were searched in the catalog and then added to the shopping cart, addresses and payment methods were added, and the entire purchase flow was tested. All components generated logs, which were later used for training.

To ensure consistency and allow rapid large-scale experimentation, we use a Continuous Delivery and Continuous Integration tool that automates all tests through a pipeline. CircleCI[7] is a platform that automates the software development process. It helps teams to automate testing and deployment of applications, facilitating the integration of code changes and ensuring the delivery of high-quality software.

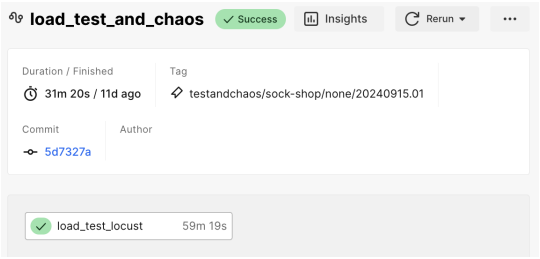

FIGURE 4. CircleCI - Locust Load Test.

For both workflows - running load testes with Locust and injection errors into the application using Chaos Mesh, as shown in Figure 4 we used the same tool. The pipeline is parameterized through GitHub TAG, where we define which microservice should be affected and the type of error to be injected. In this case, a network error was introduced into the shipping microservice.

A. LLM TRAINING

The logs employed for training are unfiltered and devoid of any predefined schema or structure. This absence of constraints allows the anomaly detection system to remain agnostic to the specific log ingestion agents employed.

[7]https://circleci.com/

Log data collection is facilitated through daemon sets deployed across all nodes within the Elastic Kubernetes Service (EKS) cluster. These agents interface with the applications' standard output **(stdout)**, gather the emitted data, and transmit it to an in-memory buffer within the Grafana Loki application.

At one-minute intervals, this data is flushed and written to an Amazon S3 bucket. This strategy significantly reduces storage costs compared to maintaining the data in Persistent Volume Claims (PVC) within the Kubernetes environment.

However, we observed that incorporating this metadata during model training could degrade performance. Much of the metadata is static, offering limited value and potentially introducing noise that interferes with detecting meaningful relationships between events. To align with our goal of working with unstructured, schema-free data in its raw form, we implemented a pre-processing step to remove unnecessary metadata, retaining only the events crucial for training.

The log anomaly detection module consists of an LLM, specifically Llama-3 8b [22], and a sentence similarity model [23]. By fine-tuning Llama-3 8b on logs collected under the normal system execution, the model learns and adapts to the expected log patterns of the system's microservices, as shown in Figure 5.

The fine-tuned LLM then generates the next log message based on a sequence of 10 previously registered logs. Since it is trained on normal execution logs, it predicts the log message that would typically appear under normal conditions. If the actual logs differ from the ones generated by the LLM, an anomaly is detected. Instead of comparing individual log messages, the similarity between the sequences of logs generated by the LLM and the actual registered logs is computed using the sentence similarity model. If the similarity score falls below 0.8, the log sequence is classified as anomalous. After extensive hyper-parameter tuning, we selected a 0.8 threshold because anything lower made the model overly sensitive and flooded us with false positives, while anything higher turned it too conservative and let true errors slip through; 0.8 therefore offers the optimal trade-off, catching most anomalies without an exponential rise in false alarms. The fine-tuned LLM then generates the next log message based on a sequence of 10 previously registered logs. Since it is trained on normal execution logs, it predicts the log message that would typically appear under normal conditions. If the actual logs differ from the ones generated by the LLM, an anomaly is detected. Instead of comparing individual log messages, the similarity between the sequences of logs generated by the LLM and the actual registered logs is computed using the sentence similarity model. If the similarity score falls below 0.8, the log sequence is classified as anomalous. Following comprehensive hyper-parameter optimization, a decision threshold of 0.8 was adopted: values <0.8 markedly increased the incidence of false positives, whereas values > 0.8 elevated the false-negative rate by allowing genuine anomalies to go undetected. Consequently,

0.8 represents the optimal balance between sensitivity and specificity for the detection task.

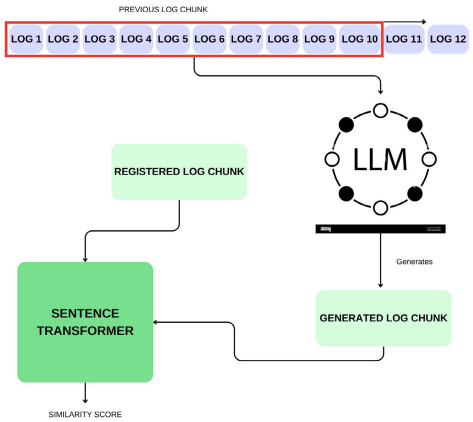

FIGURE 5. LLM training workflow.

TABLE 3. Chaos Mesh Experiments for Sock-Shop Namespace.

Service Name	Chaos Mesh Error
user	Pod Failure
session-db	Network Partition (disconnected from FE)
front-end	Pod Failure
carts-db	IO Failure (50% error rate)
catalogue-db	IO Failure (50% error rate)
payment	Pod Failure (one instance)
rabbitmq	Network Partition (disconnected from queue)
shipping	Bandwidth Limit (200kbps)
shipping	Pod Failure
shipping	HTTP Abort (on GET request to /, port: 80)
shipping	Memory Stress (512MB, 4 workers)
shipping	Network Delay (200ms latency)

Additional layers were incorporated, followed by load testing using Locust to simulate standard purchase flows within the sock shop environment.

The CircleCI pipeline ran for approximately one hour, during which all microservices were invoked in the tests.

Chaos engineering experiments aim to uncover potential weaknesses in distributed systems by simulating failure conditions in a controlled environment. The following list details the fault injection scenarios applied to various microservices using Chaos Mesh. Each experiment targets a specific failure mode, such as pod failures, network partitions, or resource constraints, to evaluate the system's resilience, error handling, and recovery mechanisms. The explanations provide context for the purpose of each scenario, its implications on the system, and the expected outcomes.

By generating this workflow repeatedly in testing scenarios, the model was exposed to a wide variety of user interactions. This process allowed us to evaluate system performance, identify bottlenecks, and fine-tune the model to handle real-world purchase flows effectively. The Figure 6 provides a Top-Down Overview about the load test workflow.

The blue components represent randomized flows, where multiple actions, such as adding and removing payment methods, addresses, or catalog searches, are performed to

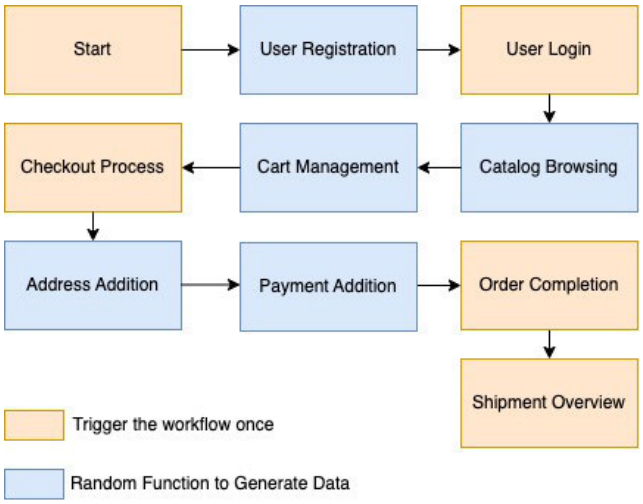

FIGURE 6. Locust Load Test Workflow.

simulate realistic user behaviors. In contrast, the yellow components represent sequential flows that occur only once per testing round, progressing from the beginning to the end of the workflow. For each load test conducted to train the model, hundreds of iterations following this workflow were executed repeatedly, generating a substantial volume of logs to support model training. This methodology ensured the activation of all microservices listed in Table 2 through various interactions, resulting in the generation of event logs.

The logs employed for training are unfiltered and devoid of any predefined schema or structure. This absence of constraints allows the anomaly detection system to remain agnostic to the specific log ingestion agents employed.

Log data collection is facilitated through daemon sets deployed across all nodes within the Elastic Kubernetes Service (EKS) cluster. These agents interface with the applications' standard output **(stdout)**, gather the emitted

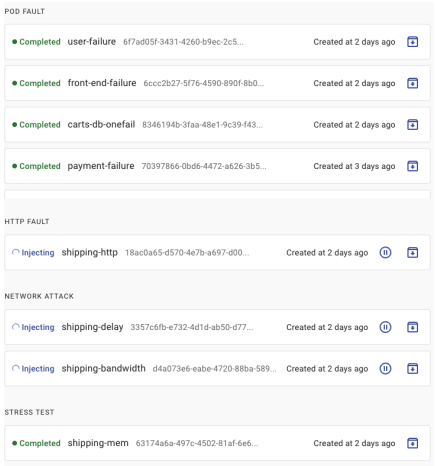

FIGURE 7. Chaos Mesh Console UI.

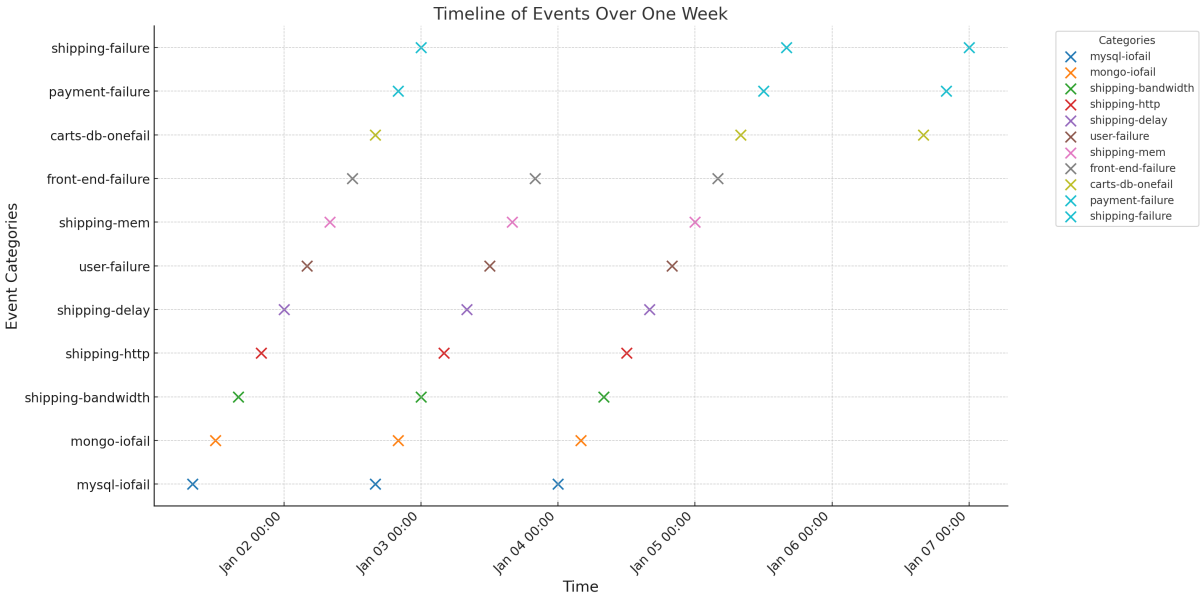

FIGURE 8. Chaos Mesh experiment timeline.

data, and transmit it to an in-memory buffer within the Grafana Loki application.

At one-minute intervals, this data is flushed and written to an Amazon S3 bucket. This strategy significantly reduces storage costs compared to maintaining the data in Persistent Volume Claims (PVC) within the Kubernetes environment.

The errors introduced in each microservice during the Chaos Mesh experiment are summarized in Table 3. The duration of these disruptions varied, averaging around 10 minutes. The entire test spanned 30 minutes, with the first 10 minutes serving as a baseline period under normal system load, free from any injected faults. During the next 10 minutes, the faults listed in Table 1 were introduced. The final 10 minutes mirrored the initial baseline, with no further errors injected, allowing the system to return to normal operating conditions.

Using the Chaos-mesh graphical interface we can follow and interact with the tests in real time. As shown in Figure 7 tests logs and all events are displayed by the tool.

Figure 8 Illustrates the chronological sequence of events in a Chaos Mesh experiment using a timeline.

FIGURE 9. Log Chunk Registered.

The **Registered Block** in Figure 9 refers to the set of logs captured during actual system operations, while the **Generated Block** in Figure 10 represents the logs predicted by the model. An anomaly is identified when the similarity score (SIM SCORE) between the **Registered Block** and the **Generated Block** falls below a predefined threshold of 0.8. Specifically, if the cosine similarity drops below 0.8, the deviation is considered substantial enough to be classified as an anomaly.

FIGURE 10. Log Chunk Generated.

Cosine similarity is a measure used to determine the similarity between two non-zero vectors in an inner product space, often used in fields like information retrieval, machine learning, and natural language processing. It calculates the cosine of the angle between two vectors, providing a metric of orientation rather than magnitude.

The cosine similarity between two vectors $\mathbf{A}$ and $\mathbf{B}$ is defined as:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Where:

- $\mathbf{A} \cdot \mathbf{B}$ is the dot product of vectors $\mathbf{A}$ and $\mathbf{B}$.
- $\|\mathbf{A}\|$ is the magnitude (or norm) of vector $\mathbf{A}$, calculated as $\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{n} A_i^2}$.
- $\|\mathbf{B}\|$ is the magnitude (or norm) of vector $\mathbf{B}$, calculated as $\|\mathbf{B}\| = \sqrt{\sum_{i=1}^{n} B_i^2}$.

The resulting value of cosine similarity ranges from -1 to 1:

- A value of 1 indicates that the two vectors are identical in direction.
- A value of 0 indicates orthogonality, meaning there is no similarity between the vectors.
- A value of -1 indicates that the two vectors are diametrically opposed, pointing in opposite directions.

For the example shown in Figure 10, the SIM SCORE result is 0.82. This value is close to 1, indicating high similarity between the registered logs and the generated logs. Consequently, this suggests a low probability of the logs being anomalous, as the similarity score exceeds the predefined threshold of 0.8. This result supports the conclusion that the log patterns align well with expected behavior, reinforcing the model's effectiveness in detecting anomalies.

Given that the volume of logs generated amounted to several thousand lines for both Registered and Generated events, we manually classified a representative sample of approximately 10% of the total dataset. This classification serves as the foundation for generating the results and categorizing the predictions within the model, which will be elaborated upon in the subsequent chapter.

This manual classification allows for a more accurate assessment of the model's performance and provides valuable insights into the effectiveness of the anomaly detection process. By establishing a ground truth through this sample, we can better evaluate the model's predictions and refine its capabilities in identifying anomalies in log data.

B. BAYNET WORKFLOW

The proposed system, Baynet, leverages the Python library **pgmpy**, which includes support for dynamic Bayesian networks. Given the highly dynamic relationships among system components, Baynet adopts a flexible architecture that allows real-time updates of values and weights within the Bayesian network, reflecting the evolving nature of events.

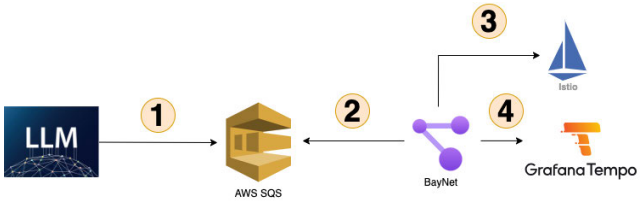

FIGURE 11. Baynet Architecture.

In the following sections, we detail the operational dynamics of Baynet and its architectural components:

1) **Input Data Handling:**
   Baynet ingests data in the form of JSON payloads derived from large language model (LLM) training outputs. Upon detecting one or more anomalies, this data is queued asynchronously via an Amazon Simple Queue Service. Each message sent to the queue encapsulates critical metadata, including the name of the affected service, the type of anomaly detected, timestamps, and additional contextual information.
2) **Trace Analysis:**
   The system initiates an in-depth trace analysis for microservices flagged with anomalies. This step ensures that only traces associated with the affected components are subjected to scrutiny, optimizing computational efficiency.
3) **Dynamic Weight Adjustment:**
   Based on the trace analysis, Baynet adjusts the weights of its Bayesian network to reflect the presence of errors identified within spans or traces. These modifications enhance the network's representational accuracy in real time.
4) **Component Relationship Modeling:**
   Baynet incorporates dynamically adjustable relationships between system components. By leveraging data provided by Istio, the framework establishes and updates the interconnections among microservices, ensuring a comprehensive and adaptive representation of system dependencies.

Through this dynamic inference process, Baynet delivers Bayesian analyses tailored to the transient behaviors inherent in distributed systems.

The initial phase of the workflow involves a comprehensive analysis of all Grafana Tempo[8] traces, as illustrated in the accompanying Figure 12.

Grafana Tempo is an open-source, distributed tracing system designed to seamlessly integrate with the Grafana observability stack. It allows developers and operators to collect, store, and query traces from distributed systems. Tempo's primary focus is to provide scalable, cost-efficient, and simplified tracing without the need for complex storage systems, as it is optimized for high-throughput environments.

Most recent traces

Trace ID	Start time	Service	Name	Duration
5ceb8cfe6002e2ed9d18f...	2024-11-18 20:26:48	catalogue.sock-shop	catalogue.sock-shop.svc.cluster.local:80/*	1 ms
b854a0d13b59809ed7b...	2024-11-18 20:25:57	payment.sock-shop	payment.sock-shop.svc.cluster.local:80/*	1 ms
a742c79a1c8e8e92bccc1...	2024-11-18 20:24:18	catalogue.sock-shop	catalogue.sock-shop.svc.cluster.local:80/*	1 ms
8050a37154f5fec210a98...	2024-11-18 20:23:40	rabbitmq.sock-shop	:0/*	
64dd54f7f65fe79fbb3814...	2024-11-18 20:18:42	rabbitmq.sock-shop	rabbitmq.sock-shop.svc.cluster.local:80/*	10 ms
2e452f1bdc7851388d90f...	2024-11-18 20:18:40	rabbitmq.sock-shop	:0/*	
a0522acdb372ee88eae9...	2024-11-18 20:17:57	payment.sock-shop	payment.sock-shop.svc.cluster.local:80/*	1 ms
a3595ad1319639ba6cfaf...	2024-11-18 20:16:53	queue-master.sock-shop	queue-master.sock-shop.svc.cluster.local:80/*	

FIGURE 12. Trace ID from Tempo Data source.

Interactively, Grafana enables us to monitor the duration, errors, and various other metadata extracted from these

[8]https://grafana.com/oss/tempo/

traces, as depicted in the Figure 13. The Tempo data is derived from Istio's network information, collected through the Envoy proxies, and supplemented with additional data from the application itself. This combination provides a comprehensive data source that offers visibility at a high level, capturing insights from the perspective of inter-service calls across all microservices.

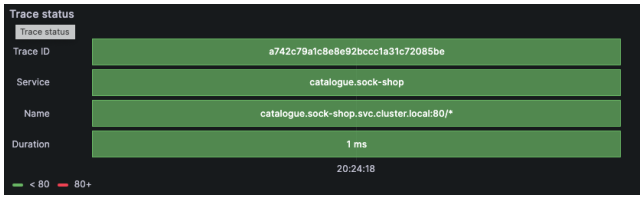

FIGURE 13. Traces Status from Tempo Data Source.

Based on the collected traces, we begin by cleaning the data and removing irrelevant metadata for Bayesian inference.

The **get_score** function processes service trace data and calculates a reliability score for each service based on several performance metrics. The scoring system takes into account request duration, success rate, response times (P99), HTTP status codes, and recent deployment conditions.

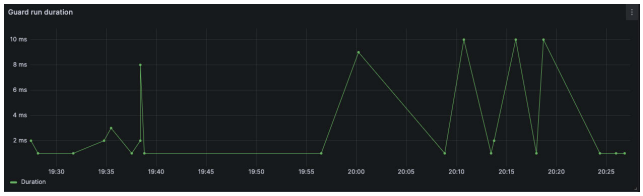

FIGURE 14. Trace ID duration from Tempo Data source.

The second function defines various thresholds and penalties used to assess service performance. These are:

- **Duration Threshold:** If a request takes longer than 1 second (1e9 nanoseconds), a penalty is applied.
- **2xx Success Rate Threshold:** If the service's success rate (the proportion of successful 2xx HTTP responses) is below 70%, a penalty is applied.
- **P99 Threshold:** If the 99th percentile response time (`p99`) exceeds 200ms (2e8 nanoseconds), a penalty is applied.
- **HTTP Status Code Penalty:** If the service returns HTTP status codes in the 500 range (server errors), a penalty is applied.
- **Recent Deployment Penalty:** If the service has been recently deployed (indicated by the `recent_deploy` flag) and the deployment's TTL (time-to-live) is less than 60 minutes, a penalty is applied.

The function defines a function `calculate_score` that computes the reliability score for each service:

- The base score starts at 1.0 (best possible score).
- For each performance metric, if the metric exceeds its threshold or condition, a penalty is applied to reduce the score.
- The penalties for each metric are calculated as follows:
  - Duration: A penalty is applied if the duration exceeds the threshold.
  - 2xx Rate: A penalty is applied if the success rate is below the threshold.
  - P99: A penalty is applied if the P99 response time exceeds the threshold.
  - Status Codes: A penalty is applied for any 500 series HTTP status code.
  - Recent Deployment: A penalty is applied if the deployment TTL is less than 60 minutes.
- The final score is clamped between 0 and 1, ensuring that no score exceeds 1 or falls below 0.

The function processes the input data as follows:

- It reads the file `preprocessed_traces.csv` into a DataFrame.
- The `calculate_score` function is applied to each row to compute the score for each service.

The function creates a new DataFrame with the calculated scores, containing only the service names and their respective scores. This DataFrame is saved to a new CSV file called `service_scores.csv`.

Although the function includes a note about dynamically retrieving thresholds from Prometheus and Istio. Finally, the function evaluates the health and reliability of services based on several performance metrics and generates a score for each service. The results are stored in a CSV file for further analysis.

Lastly, we built a Bayesian Network to model the failure probabilities of microservices based on their health scores. It loads service relationships and health scores from CSV files, builds a network where services are nodes connected by edges representing dependencies, and defines Conditional Probability Distributions (CPDs) for each service based on its health score. Using Variable Elimination, it calculates the probability of each service being down and stores the results in both a CSV file and Redis. This model helps predict service failures and can be used for automated monitoring or decision-making in a microservices architecture.

The final result is displayed using a custom plugin NodeGraph API[9] and can be represented using Figure 15. For each microservice that exists in the system, a node was created, and the relationships and connections between them are represented by Edges.

Within each node there is the probability of outage, for that period of time, according to the Bayesian inference generated in the previous step. It is worth remembering that the values are dynamic, as are the relationships of other services. If a new microservice is added or removed, it will be automatically updated on the dashboard.

[9]https://grafana.com/grafana/plugins/hamedkarbasi93-nodegraphapi-datasource/

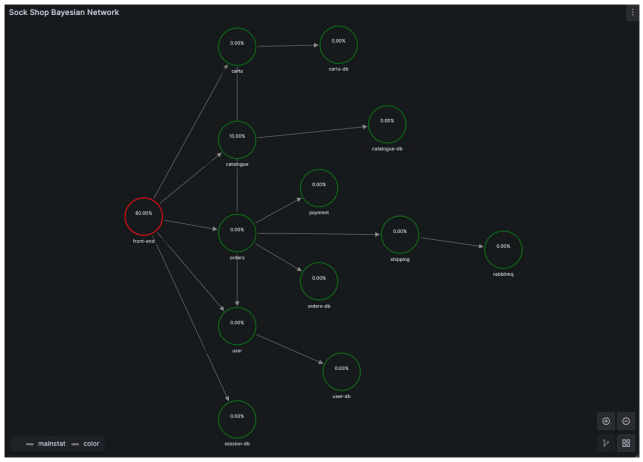

FIGURE 15. Bayesian Network using NodeGraph.

VI. OUTCOMES AND DISCUSSIONS

The Figure 16 heatmap visualizes the performance of several anomaly detection models across different test scenarios (represented by filenames). Each row corresponds to a test, and each column represents a performance metric: Accuracy, Precision, Recall, and F1-score. The color intensity corresponds to the metric's value, with darker blues indicating higher values and lighter yellows indicating lower values.

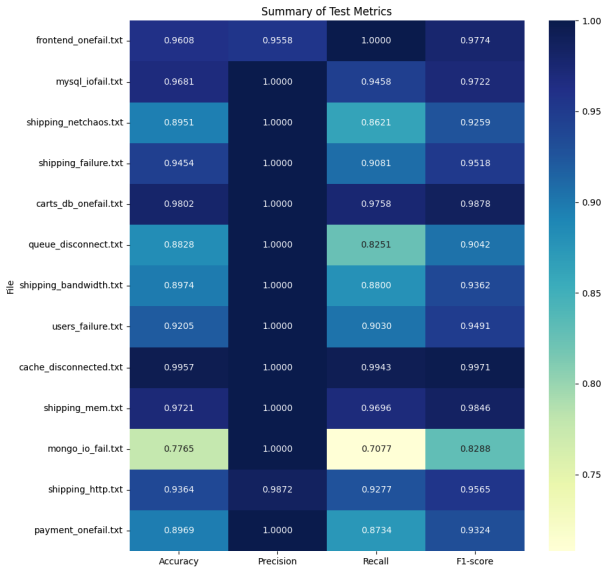

FIGURE 16. LLM Heatmap Matrix.

- **High Precision:** Across almost all tests, the precision is very high (often 1.00 or very close). This means that when the model predicts an anomaly, it is almost always correct. This is a positive sign, indicating few false alarms.
- **Varied Recall:** The recall varies more significantly between tests. This indicates that the models have different abilities to capture all actual anomalies. Some tests have near-perfect recall (e.g., `Frontend Pod Failure`), while others have lower recall.
- **Accuracy and F1-Score Follow Recall:** The accuracy and F1-score generally follow the trend of the recall. When recall is high, accuracy and F1-score are also high. This is expected, as recall plays a significant role in both metrics.
- **False Negatives are the Primary Issue:** The main source of errors seems to be false negatives (missing actual anomalies), as indicated by the varying recall values and the fact that false positives are low in most tests.

The heatmap effectively highlights that while the models generally have high precision (few false positives), their ability to detect all actual anomalies (recall) varies across different test scenarios. The *Mongo IO Failure* test is a clear outlier with significantly lower performance, and the near-perfect performance on *Cache Disconnected* warrants further investigation to rule out overfitting. The heatmap provides a quick and intuitive way to compare the models' performance across different tests and identify areas for potential improvement.

- `Cache Disconnected Failure`: This test shows perfect performance (1.00 for all metrics). This suggests the model is very effective at detecting this specific type of anomaly. However, perfect performance on a single test could indicate overfitting and should be examined with additional data.
- `Mongo IO Failure`: This test has the lowest recall (0.71) and consequently the lowest accuracy (0.78) and F1-score (0.83). This indicates the model struggles to detect this type of anomaly. Further investigation is needed to understand why and improve performance on this specific scenario.
- `Queue Network Disconnect Failure`: This test also has a relatively lower recall (0.83) and a higher number of false negatives (64) compared to other tests, indicating room for improvement in detecting queue disconnections.
- `Shipping Netchaos Failure`: This test shows good performance with high accuracy (around 0.90-0.95), high precision (1.00), and reasonably good recall (0.86-0.91).
- `Shipping Pod Failure`: This test shows good performance with high accuracy (around 0.90-0.95), high precision (1.00), and reasonably good recall (0.86-0.91).
- `User Pod Failure`: This test shows good performance with high accuracy (around 0.90-0.95), high precision (1.00), and reasonably good recall (0.86-0.91).
- `Shipping Bandwidth Failure`: This test shows good performance with high accuracy (around 0.90-0.95), high precision (1.00), and reasonably good recall (0.86-0.91).
- `Shipping HTTP Failure`: This test is unique in having a small number of false positives (2). While the

number is low, it's worth investigating why these errors occurred.

- **Other Tests:** (`Frontend Pod Failure`, `MySQL Failure`, `Cards DB Pod Failure`, `Payment Pod Failure`, `Shipping Memory Failure`): These tests show very high performance with accuracy, precision, recall, and F1-score generally above 0.95, indicating the models are very effective in these specific scenarios.

A. LLM KEY TAKEAWAYS

- **Focus on Improving Recall:** The main area for improvement is increasing recall, especially for tests like `Mongo IO Failure` and `Queue Disconnect Failure`. This could involve:

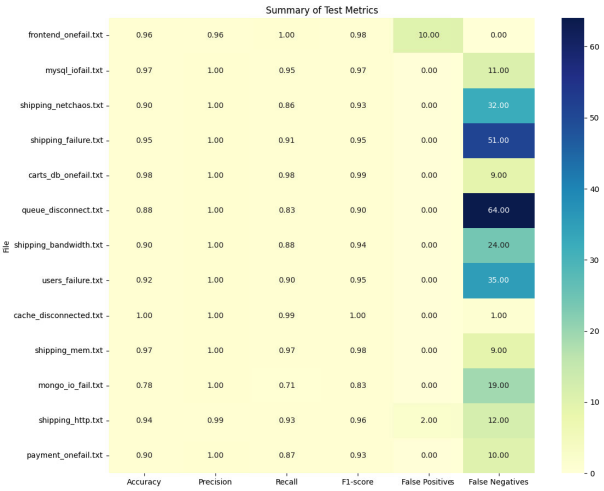

FIGURE 17. LLM Summary Results.

  - Collecting more data for these specific anomaly types.
  - Adjusting model parameters or thresholds to be more sensitive to anomalies.
  - Exploring different modeling techniques that might be better suited for these types of anomalies.
- **Investigate False Positives in `Shipping HTTP Failure`:** While the number of false positives is low, understanding why they occurred can help further refine the model.
- **Validate `Cache Disconnected Failure` Performance:** The perfect performance on this test should be validated with additional data to rule out overfitting.
- **Consider Cost of Errors:** Depending on the application, the cost of false negatives (missing a real anomaly) might be higher than the cost of false positives (false alarms). This should be considered when deciding on the desired trade-off between precision and recall.

Figure 17 shows a TOP-Down approach with all data from all tests summarized in a single table, to facilitate comparison and visualization of outliers.

B. BAYESIAN NETWORK PREDICTION RESULTS

In this section, we present the results of the Bayesian network predictions, highlighting its performance and accuracy in detecting and classifying anomalies. The analysis is based on the errors approach used during the anomaly identification phase, and the workflow integrates the Bayesian network with trace analysis for a comprehensive evaluation. Key metrics, comparative insights, and practical implications of the prediction outcomes are discussed to validate the efficacy of the approach. Based on the probabilities derived from the Bayesian network, the corresponding levels are represented using the following color coding:

- 0-15% - Green (low outage probability)
- 16-45% - Yellow (moderate outage probability)
- 46-99% - Red (high outage probability)

A snapshot[10] was generated with tests of some experiments so that it is possible to visualize the dynamics and how the dashboards are displayed at the time of the test. Some data may not be available. Raintank is the one who stores and makes the data available through these links in the footer.

For the performance evaluation of the Bayesian network, we utilized the same error (3) employed during the anomaly identification and classification phase. For each anomaly detected by the LLM, we initiated the trace analysis workflow, followed by the construction of the Bayesian network.

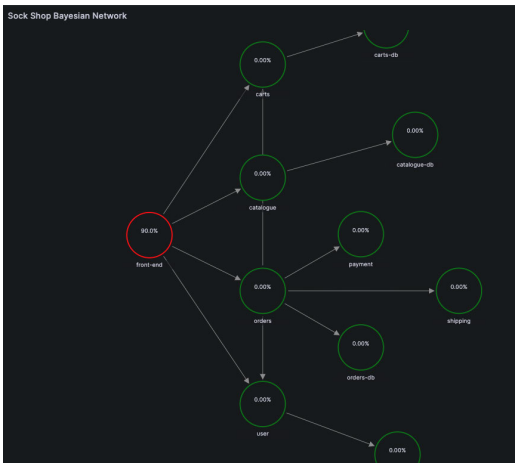

FIGURE 18. Bayesian Network Performance Analysis - Frontend.

The Figure 18 demonstrates the relationships and dependencies among various components of the system. The highlighted node front-end, marked in red with a 90.0% likelihood, indicates a significantly high probability of being the root cause of an anomaly.

All other nodes show a 0.00% likelihood, suggesting that no anomalies have propagated to these components or that they are not currently impacted. The network effectively outlines the causal relationships, helping identify the source of an issue and tracing its potential impact through the

[10]https://snapshots.raintank.io/dashboard/snapshot/erkWnoyE2om7UZhQksIdtUoJoA8oleF4?orgId=0

system. This test highlights the utility of Bayesian networks in pinpointing anomalies and their dependencies, providing a structured approach to troubleshooting. When the value is close to zero, it typically indicates that no traces were intercepted for that service, resulting in an outage probability near zero. In most cases where the value is zero, it suggests that the microservice was unaffected by the test, leaving no errors for the Bayesian network to analyze.

The Figure 19 identified the **shipping** microservice as the most likely point of failure, with a 78% probability of being out of order. Additionally, the **orders** microservice exhibited a 12.16% probability of failure, attributed to its dependency on the shipping microservice. This highlights the cascading effects of failures within a dependency chain: when a dependent microservice experiences an issue, the connected services may display mild symptoms of failure. However, these symptoms often remain insufficiently significant to classify the dependent microservices as failing, leading the network to still consider them operational.

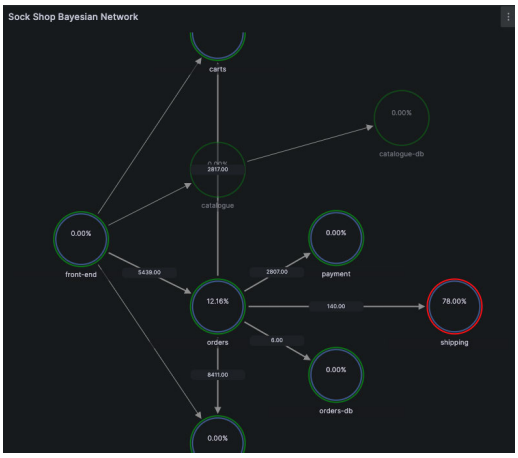

FIGURE 19. Bayesian Network Performance Analysis - Shipping.

VII. CONCLUSION AND FUTURE DIRECTIONS

The model was originally designed as a threshold-based anomaly detector trained using self-supervised learning, but it can be easily adapted into a classifier. Once the model is deployed, operators can collect and label anomaly log samples, allowing for the replacement of the model's language modeling head with a classification head to fulfill this purpose. This adaptability enhances its functionality, enabling it to categorize logs based on their anomaly status. Moreover, the model can be pre-trained on diverse log datasets, which broadens its applicability and makes it a versatile foundation for various natural language processing (NLP) tasks and log anomaly detection scenarios. This flexibility allows it to be fine-tuned for specific use cases, improving its effectiveness in detecting anomalies in different environments and with varying log characteristics.

Chaos Mesh provides a comprehensive set of error types that can be introduced into a system, making it a valuable tool for resilience testing.

As part of our future work, we plan to:

- **Map All Error Relationships**: We will systematically document how different error types affect various microservices. This will involve analyzing the impact of each error type on system performance, response times, and overall behavior.
- **Conduct Systematic Experiments**: We will perform experiments that apply each error type across all microservices in the sock shop application. By doing this, we can identify patterns of interactions between errors and understand their cumulative effects on system behavior.
- **Enhance Model Training**: The insights gained from the mapping and experiments will be used to train our anomaly detection model more effectively. By incorporating data from a wider range of error scenarios, we can improve the model's accuracy and robustness in identifying anomalies.
- **Evaluate and Iterate**: We will continuously evaluate the model's performance based on its ability to detect anomalies resulting from various error types. This iterative process will help refine the model and enhance its predictive capabilities.

By exploring these avenues, we aim to develop a more resilient system capable of adapting to and recovering from diverse failure scenarios, ultimately improving overall system stability and performance.

In certain sections of the log, numerical data appears as IP addresses, response times, status codes, and other metrics. While the LLM approach often neglects these numerically significant values, Long Short-Term Memory (LSTM) networks may provide a more effective solution.

LSTM networks are designed to handle sequential data and can retain relevant information over long sequences, making them well-suited for tasks that involve time-series data or event logs. Unlike LLMs, which may treat numerical data as mere tokens and fail to account for their contextual importance, LSTMs can learn patterns in numerical sequences, thereby incorporating this data meaningfully into the analysis.

By leveraging the strengths of LSTMs, it is possible to maintain the significance of numerical metrics while simultaneously modeling the temporal dependencies inherent in log data. This dual capability allows for a more comprehensive understanding of the context surrounding each event, potentially enhancing predictive accuracy and insight generation compared to relying solely on LLMs.

ACKNOWLEDGMENT

The authors would like to express their gratitude to Mindera for their sponsorship. Additionally, they extend their thanks to Amazon Web Services (AWS) for their generous support of this project, conducted in collaboration with the Laboratory of Distributed Systems and Concurrent Programming, Institute of Mathematical and Computer Sciences University of São Paulo (ICMC/USP). For open access purposes, they

REFERENCES

[1] W. Pourmajidi, J. Steinbacher, T. Erwin, and A. Miranskyy, "On challenges of cloud monitoring," 2018, *arXiv:1806.05914*.
[2] S. J. Fowler, *Production-ready Microservices: Building Standardized Systems Across an Engineering Organization*. CA, USA: O'Reilly Media, 2016.
[3] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. CA, USA: O'Reilly Media, 2016.
[4] R. Xu and K. Ding, "Large language models for anomaly and out-of-distribution detection: A survey," 2024, *arXiv:2409.01980*.
[5] A. Russell-Gilbert, A. Sommers, A. Thompson, L. Cummins, S. Mittal, S. Rahimi, M. Seale, J. Jaboure, T. Arnold, and J. Church, "AAD-LLM: Adaptive anomaly detection using large language models," in *Proc. IEEE Int. Conf. Big Data (BigData)*, Dec. 2024, pp. 4194–4203.
[6] S. Yang, S. Liu, P. Shang, and H. Wang, "An overview of methods of industrial anomaly detection," in *Proc. 7th Int. Conf. Robot., Control Autom. Eng. (RCAE)*, Oct. 2024, pp. 603–607.
[7] H. Jin, G. Papadimitriou, K. Raghavan, P. Zuk, P. Balaprakash, C. Wang, A. Mandal, and E. Deelman, "Large language models for anomaly detection in computational workflows: From supervised fine-tuning to in-context learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2024, pp. 1–17.
[8] J. Liu, C. Zhang, J. Qian, M. Ma, S. Qin, C. Bansal, Q. Lin, S. Rajmohan, and D. Zhang, "Large language models can deliver accurate and interpretable time series anomaly detection," 2024, *arXiv:2405.15370*.
[9] W. Guan, J. Cao, S. Qian, J. Gao, and C. Ouyang, "LogLLM: Log-based anomaly detection using large language models," 2024, *arXiv:2411.08561*.
[10] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "LogFiT: Log anomaly detection using fine-tuned language models," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 2, pp. 1715–1723, Apr. 2024.
[11] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.
[12] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2021, pp. 492–504.
[13] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–14.
[14] A. Li, Y. Zhao, C. Qiu, M. Kloft, P. Smyth, M. Rudolph, and S. Mandt, "Anomaly detection of tabular data using LLMs," 2024, *arXiv:2406.16308*.
[15] S. Gamboa and E. Orduña, "Feasibility of distributed monitoring and distributed control in power system," in *Proc. IEEE PES Innov. Smart Grid Technol. Conf. Latin Amer.*, Sep. 2017, pp. 1–6.
[16] L. E. G. Pulcinelli, D. F. Pedroso, and S. M. Bruschi, "Conceptual and comparative analysis of application metrics in microservices," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput. Workshops (SBAC-PADW)*, Oct. 2023, pp. 123–130.
[17] X. Jiang, J. Li, H. Deng, Y. Liu, B. B. Gao, Y. Zhou, J. Li, C. Wang, and F. Zheng, "MMAD: The first-ever comprehensive benchmark for multimodal large language models in industrial anomaly detection," 2024, *arXiv:2410.09453*.
[18] M. Zhang, Y. Shen, J. Yin, S. Lu, and X. Wang, "ADAGENT: Anomaly detection agent with multimodal large models in adverse environments," *IEEE Access*, vol. 12, pp. 172061–172074, 2024.
[19] F. Chen, L. Zhang, G. Pang, R. Zimmermann, and S. Deng, "Facilitate collaboration between large language model and task-specific model for time series anomaly detection," 2025, *arXiv:2501.05675*.
[20] D. F. Pedroso, L. E. G. Pulcinelli, W. A. A. Aisawa, and S. M. Bruschi, "AWS powered cloud research environment PaaS," in *Proc. Anais da 15th Escola Regional de Alto Desempenho de São Paulo (ERAD-SP)*, May 2024, pp. 101–104.
[21] J. Doe and J. Smith, "A survey of container orchestration mechanisms," *Cloud Comput. Res. J.*, vol. 10, no. 3, pp. 123–145, 2021.
[22] A. Grattafiori et al., "The llama 3 herd of models," 2024, *arXiv:2407.21783*.
[23] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "MPNet: Masked and permuted pre-training for language understanding," 2020, *arXiv:2004.09297*.

DIEGO FRAZATTO PEDROSO received the degree in security information, the M.Sc. degree in network and telecommunications, and the Ph.D. degree in computer science. With more than ten years of experience in cloud computing and system scalability, he specializes in Site Reliability Engineering (SRE) and Development. He has worked with several cloud providers and has expertise in managing and optimizing high-availability, scalable, and secure systems.

LUÍS ALMEIDA received the M.Sc. degree in data science and computer science from FCUP in October 2023, and the Ph.D. degree in artificial intelligence from FCUP in February 2024. He is an Affiliated Researcher at INESC TEC, where he has been working since September 2024. Prior to this, he was an AI Technical Writer at NOS SGPS from June to September 2024. He also held positions as a Research Fellow at the USP and the FCUP between 2023 and 2024.

LUCAS EDUARDO GULKA PULCINELLI is currently pursuing the degree in computer science with ICMC. He is an MLOps Intern at Wikki Brasil, where he has been working since January 2025. His main activity involves the development of a system focused on deploying prescriptive machine learning models. Prior to this, he was a Research Assistant at the University of São Paulo from October 2021 to September 2024.

WILLIAM AKIHIRO ALVES AISAWA is currently pursuing the Ph.D. degree in computer science with ICMC. He is working as a Site Reliability Engineer (SRE) at Bradesco, where he has been working since December 2022. In his role, he is responsible for spreading SRE culture and principles across squads, ensuring technical and operational excellence.

INÊS DUTRA is currently an Assistant Professor with the Department of Computer Science, Faculty of Sciences of the University of Porto, Portugal, where she teaches mostly subjects related to artificial intelligence, machine learning, and distributed and parallel computation. Her main research interests include logic programming, (probabilistic) inductive logic programming, statistical relational learning, parallelization, and biomedical applications.

SARITA MAZZINI BRUSCHI received the bachelor's degree in computer science from UNESP, in 1994, the master's and Ph.D. degrees in computer science from University of São Paulo (USP), São Carlos, in 1997 and 2002, respectively, and the Postdoctorate degree in computing from George Washington University, in 2016. She is currently a Ph.D. Professor MS3 (level 2) RDIDP with USP and the President of the Undergraduate Committee of ICMC.