National College of Ireland

A Machine Learning-based Monitoring Framework for Prediction, Diagnosis, and Feedback Learning in DevOps

MSc Research Project
Cloud Computing

Boyang Jiang
Student ID: 23399937

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Boyang Jiang
Student ID:	23399937
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	A Machine Learning-based Monitoring Framework for Prediction, Diagnosis, and Feedback Learning in DevOps
Word Count:	4391
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Boyang Jiang
Date:	25th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Machine Learning-based Monitoring Framework for Prediction, Diagnosis, and Feedback Learning in DevOps

Boyang Jiang
23399937

Abstract

Modern application complexity often leads to faults detected only after user impact. This paper reflects on real-world experiences to identify persistent challenges such as **fault prediction, root cause diagnosis, and post-failure learning in enterprise monitoring systems**. To address these issues, we designed **machine learning monitoring framework through a closed loop based on the Google SRE's incident lifecycle model: pre-failure, in-failure, and post-failure** to enhance detection accuracy, reduce MTTR (i.e., mean time to recovery), and improve system reliability. Through a review of existing literature, we confirm that these issues are widely acknowledged across academia and industry. We further summarize the common limitations of current solutions and outline key technical challenges. Finally, this proposal outlines a hypothetical methodology that uses various data sources (e.g., Log, DevOps Pipeline) to train the model. These models are mapped to each lifecycle stage, followed by a Gantt chart outlining the implementation plan.

1 Introduction

1.1 Background

Monitoring systems remain a significant challenge in modern system architecture. Due to its cloud infrastructure, microservices, and containerized related technology, there is increasing complexity Soldani and Brogi (2022). First, **the need for proactive fault prediction** Jeyarajan et al. (2025): traditional monitoring tools usually trigger alerts only after failures occur, meaning the user experience is already affected. In contrast, machine learning-based monitoring can predict potential issues in advance, allowing for proactive maintenance. Second, **the inefficiency of manual root cause analysis** Shen et al. (2020): conventional diagnostics are time-consuming and depend heavily on engineers' expertise, prolonging MTTR. Machine learning enables automated correlation and diagnosis to accelerate recovery. Third, **the problem of alert fatigue** Teggi et al. (2022): excessive, redundant alerts often obscure real anomalies, while ML-based monitoring can intelligently aggregate and filter them. Fourth, **the lack of continuous learning from past failures**: traditional reviews are manual and periodic, whereas machine learning supports continuous improvement from historical incidents. Finally, **the inability to adapt to dynamic architectures**: in cloud-native, microservice-based

environments, static rule-based systems struggle to cope with rapid changes, while AI can dynamically learn and adapt from data patterns.

Artificial Intelligence (AI), particularly Machine Learning (ML), as its sub-discipline, has demonstrated enhancing systems or applications Alenezi et al. (2022). The DevOps industry is the same. In other words, the DevOps field is extremely needed. However, its practical integration into DevOps still has certain limitations. Particularly, machine learning integrates with monitoring systems Dang et al. (2019).

Therefore, the core research problem addressed in this study is **how to effectively integrate machine learning into DevOps monitoring systems** to enable intelligent fault prediction, rapid root cause diagnosis, and continuous improvement through feedback learning from previous failures.

1.2 Research Question Definition

Based on the previous background and motivation, this study addresses the following clearly defined and measurable research question:

How much MTTR can be reduced by a machine learning-based intelligent monitoring system compared to traditional monitoring systems in DevOps environments?

Overall Objective The overall objective of this research is to design and evaluate a machine learning-based monitoring framework capable of proactively detecting faults, rapidly diagnosing root causes, and continuously improving from historical data, thereby significantly reducing MTTR for the system or application.

Specific Sub-objectives To achieve the overall objective, we define the following specific, actionable sub-objectives:

1. Identify and evaluate the most relevant, reliable, and high-quality data sources suitable for training machine learning models in DevOps monitoring scenarios.

2. Determine effective preprocessing and feature engineering methods for handling diverse monitoring data (e.g., logs, metrics, pipeline events).

3. Implement a model-serving deployment strategy that integrates seamlessly with existing DevOps monitoring workflows and enables real-time prediction and diagnosis.

4. Design and conduct experiments to empirically evaluate and compare the MTTR reduction effectiveness between traditional monitoring and the proposed intelligent machine learning-based monitoring framework.

1.3 Contribution of Research

The potential contributions of this research can be outlined from three main perspectives.

Contributions to the DevOps Field: This research leverages integrated machine learning into the DevOps field to explore AIOps practices. It provides insights and valuable experiences for DevOps engineers who tend to practice AIOps. And potentially enables further innovations such as intelligent CI/CD pipeline management, configuration management, and predictive resource management (e.g., cost prediction).

Contributions to Real-world Enterprises: By effectively reducing MTTR, our proposed intelligent monitoring framework can significantly enhance application availability and reliability. It helps enterprises transition smoothly from traditional reactive monitoring approaches to proactive, predictive, and intelligent monitoring, ultimately minimizing the number of failures and improving overall business continuity.

Contributions to Academic Research: This study addresses and fills a practical knowledge gap concerning the integration of artificial intelligence with DevOps monitoring practices. Specifically, it introduces a clear and measurable evaluation method using MTTR to assess and validate the superiority of intelligent monitoring solutions over traditional methods. Thus, the research enriches the existing literature by providing empirical evidence and structured methodologies for future studies in the AIOps area.

1.4 Structure of the Document

The remainder of this dissertation is structured as follows: **Section 2** reviews prior research on AI-driven DevOps, data quality management, and machine learning–based monitoring, establishing the theoretical foundation of this study. **Section 3** outlines the research methodology, including how the proposed solution was conceptualized and developed, the industry practices and methodologies referenced, and the evaluation strategy used to compare the performance of our framework with conventional monitoring systems. **Section 4 and Section 5** present the system design and layer-based implementation details of the intelligent monitoring framework, covering data collection, data preprocessing, model development, and deployment. **Section 6** reports the experimental results, including three case studies, and discusses key findings, limitations, and their implications. Finally, **Section 7** concludes the study and outlines directions for future research.

2 Related Work

2.1 AI and Machine Learning in DevOps

The rapid evolution of enterprise IT operations toward data-driven decision-making has accelerated the integration of artificial intelligence (AI) into traditional DevOps practices, marking the shift toward the emerging field of AIOps. Alenezi et al. (2022) demonstrated that integrating machine learning can enhance DevOps efficiency, yet their study remained theoretical and lacked practical validation. In contrast, Ali and Puri (2024) implemented AI solutions in real-world environments and reported measurable operational gains, but failed to provide methodological transparency, limiting reproducibility.

Several works explored machine learning frameworks for intelligent monitoring. Abbas and Garg (2024) acknowledged that success strongly depends on high-quality data sources. Valli et al. (2023) proposed a deep learning–based system for fault prediction and diagnosis, demonstrating promising accuracy yet omitting key details on data preprocessing and model evaluation. Hany Fawzy et al. (2023) designed an anomaly detection framework embedded in CI/CD pipelines, but its reliance on idealized environments limited real-world applicability. Meanwhile, Thantharate (2023) and Shen et al. (2020) confirmed that AI monitoring can enhance observability and self-healing capacity but offered no comparative analysis with traditional systems. Jeyarajan et al. (2025) filled part of this gap by evaluating multiple models across reliability metrics, though without implementation details on ML deployment.

2.2 Limitations of Existing Approaches

In summary, while these studies collectively validate the potential of AI-enhanced DevOps, they share several limitations: (1) a lack of empirical, reproducible experiments; (2) insufficient integration of diverse data sources such as metrics, logs, and commits; and (3) the absence of standardized evaluation frameworks. Therefore, this research aims to design an end-to-end intelligent monitoring framework that integrates fault prediction, diagnosis, and feedback learning, bridging the gap between theoretical AIOps models and their practical deployment in DevOps environments.

Table 1 Comparison of Selected Research Papers.

Table 1: Comparison of Selected Research Papers

Paper	**Methodology**	**Dataset**	**Evaluation Metrics**	**Reported Results**	**Implementation Description**
Alenezi et al. (2022)	Qualitative review	Secondary data	✗	✗	✗
Ali and Puri (2024)	Case study	DevOps pipeline	✓	✓	✗
Abbas and Garg (2024)	Mixed(quantitative, qualitative)	Operational data	✓	✓	✗
Valli et al. (2023)	Mixed (questionnaire)	Questionnaire data	✓	✓	✗
Hany Fawzy et al. (2023)	DevOps Anomaly Detection framework	DevOps pipeline	✓	✓	✓
Thantharate (2023)	System design(IntelligentMonitor)	Simulated data	✓	✓	✗
Shen et al. (2020)	System design(Proton)	DevOps pipeline	✓	✓	✗
Jeyarajan et al. (2025)	Mixed (quantitative, qualitative)	DevOps pipeline	✓	✓	✗

3 Methodology

3.1 Research Methodology

3.1.1 Methodological Thinking Process

In this sub-section, we describe the thought process behind the formation of our research methodology in as much detail as possible. The goal is to enable readers to clearly understand **how the proposed solution was developed and how it addresses the research problem**. Additionally, this section aims to offer practical insights into how AIOps projects can be implemented, thereby contributing to filling the gap in applied research within the AIOps domain.

First, we listed all the key steps that belong to the machine learning lifecycle or the DevOps lifecycle. Then classify them according to the scope they belong to: ML or DevOps. After researching multiple methodologies, we decided to use the **AWS Well-Architected Machine Learning Lifecycle** to guide our ML phase implementation. Since it is suitable for oriented-deployment machine learning projects. Figure 1 First of Thinking Process AIOps.

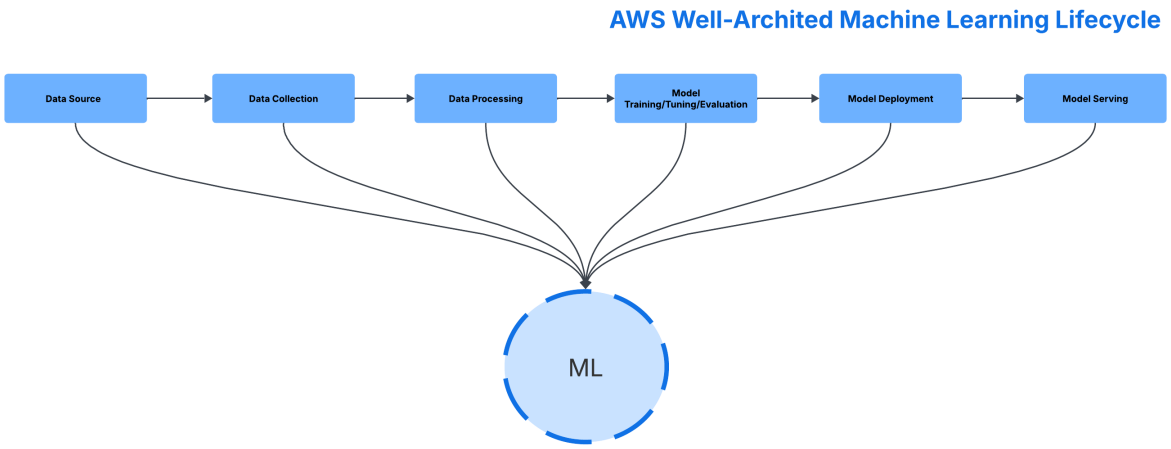

Figure 1: First of Thinking Process AIOps

For fault management in the DevOps phase, we adopted the incident management lifecycle proposed in the **Google SRE Handbook**, which segments the operational response into a closed loop of preparation, response, and learning. Figure 2 Second of Thinking Process AIOps.

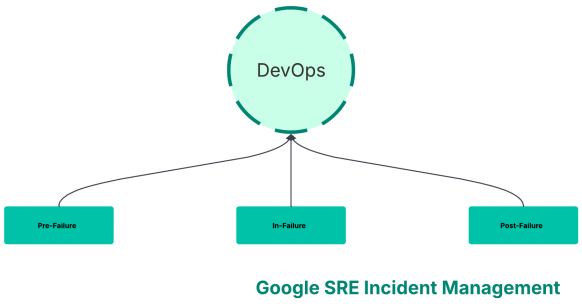

Figure 2: Second of Thinking Process AIOps

Next, we combined these two methodologies into a unified framework. In which, the model of machine learning will work at each mapped incident phase and offer inference services such as prediction, diagnosis, and feedback learning. Figure 3 Thinking Process AIOps. In summary, **the most significant challenge in implementing AIOps lies in the way of thinking.** DevOps is fundamentally a culture of collaboration and automation, and the integration of AI introduces an additional layer of complexity. Through this research, we hope to provide a structured and practical reference for applying AIOps in real-world scenarios, particularly in the context of intelligent monitoring.

3.1.2 External Methodological References

AWS Well-Architected Machine Learning Lifecycle Shrivastava et al. (2023) is proposed by AWS as a practical machine learning methodology focused on deployment processes. It consists of six main steps: business goal definition, ML problem framing, data processing, model development, model deployment, and model monitoring.

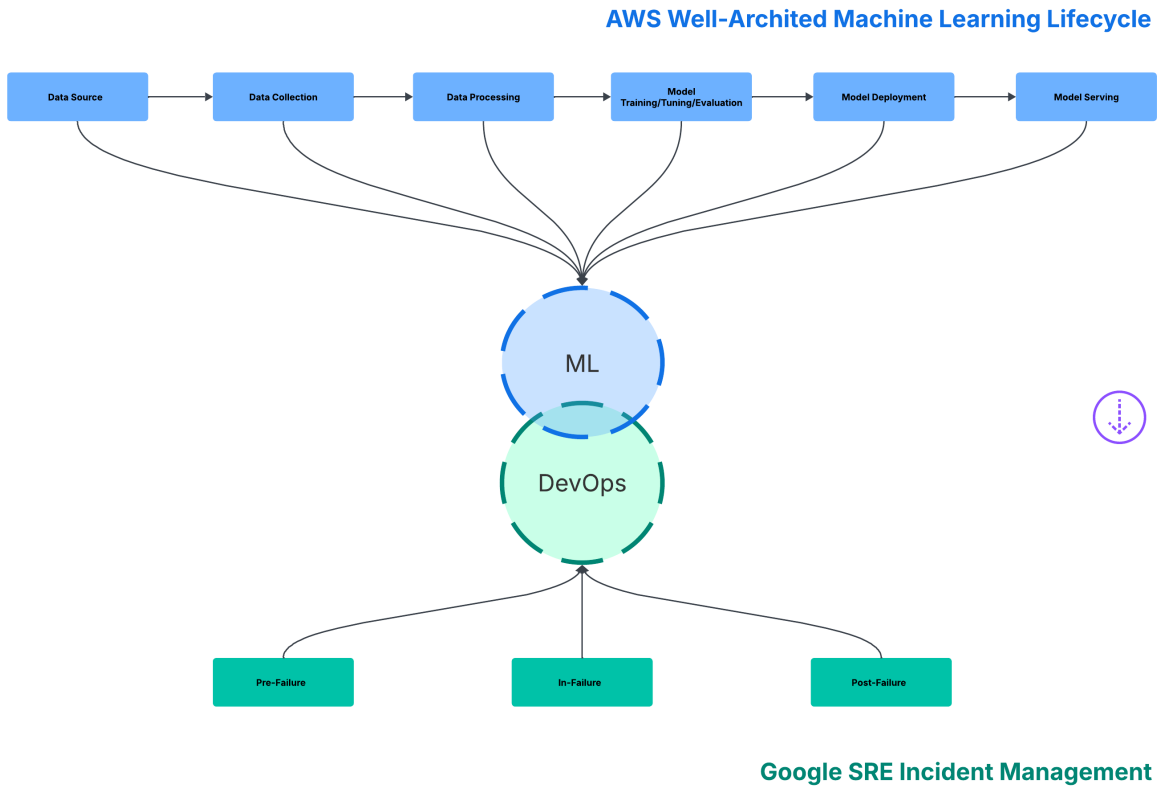

Figure 3: Thinking Process AIOps

Google SRE Incident Management Beyer et al. (2016) is proposed by the Google SRE team. The pre-failure, in-failure, and post-failure model is derived from Google's incident lifecycle, which segments incident handling into prepare, respond, and learn phases. We believe that a monitoring system should form a virtuous cycle around these three stages.

Plan–Do–Check–Act (PDCA) cycle was originally popularized by Dr. Edwards Deming as a continuous improvement methodology widely adopted in quality management and system engineering. The PDCA cycle emphasizes an iterative process of planning, implementation, evaluation, and refinement. In this research, the PDCA concept is employed to structure the overall research process.

3.2 Research Procedure

The research procedure follows the Plan–Do–Check–Act (PDCA) cycle to ensure iterative development and continuous improvement of the proposed machine learning-based monitoring framework. In each stage, we break down different tasks for better implementation. Plan: Framework Definition and Experiment Design. Do: System Implementation and Model Realization. Check: Experimental Validation. Act: Reflection and Continuous Improvement.

3.3 Evaluation Methodology

3.3.1 Why – Why Evaluate?

To determine whether our proposed intelligent monitoring system is effective or not compared to traditional monitoring. It is difficult to measure directly. Since the quality of a monitoring system is typically reflected in the monitored system. This is also a limitation that we found after our review of the literature. Most of the current papers are only measuring its self-performance. Such as how much accuracy their solution predicts, how much downtime their solution reduces Abbas and Garg (2024). We need to find a common criterion to measure the monitoring system. What value does it bring to the monitored system? Whether intelligent or traditional monitoring system.

3.3.2 How – How to Evaluate?

To evaluate the effectiveness of the intelligent monitoring system, we design a comparative experiment involving two environments: one integrated with the proposed machine learning-based monitoring system, and another using a traditional monitoring approach. Both monitoring systems are used to monitor the same application. During the experiment, we manually inject the same set of simulated faults into each environment. The evaluation is based on the monitoring response to these faults, by recording and comparing key performance indicators such as **MTTD**, **MTTR**, and **MTTResolve** across both systems. Figure 4 Evaluation Metrics.

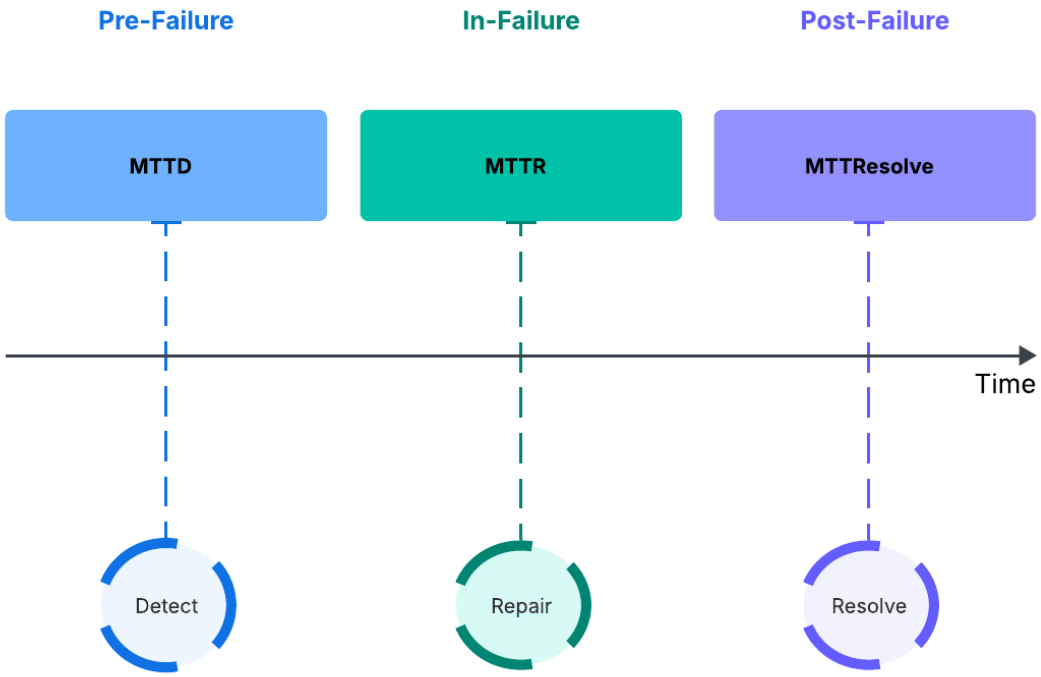

Figure 4: Evaluation Metrics

3.3.3 What – What Exactly Is Evaluated?

Since the monitoring system itself is difficult to evaluate directly, we adopt a set of time-based indicators—**MTTD (Mean Time to Detect)**, **MTTR (Mean Time to Recovery)**, and **MTTResolve (Mean Time to Resolve)**—to assess how our proposed system can help the monitored system improve availability and reliability. Specifically, MTTD reflects the system's capability to detect anomalies at an early stage, serving as a key metric in the pre-failure phase. MTTR captures the efficiency of diagnosing and repairing issues during in-failure, while MTTResolve evaluates the resolution cycle. Together, these metrics provide a comprehensive and structured approach to measuring the effectiveness of the proposed intelligent monitoring system. Table 2 Metric Selection Across Incident Lifecycle Phases.

Table 2: Metric Definitions Across Incident Lifecycle Phases

Phase	**Evaluation Metric**	**Purpose and Justification**	**Calculation Method**
Pre-Failure	MTTD (Mean Time to Detect)	Measures how early the monitoring system can detect abnormal behaviour after a fault is introduced.	DetectionTime − FaultInjectionTime
In-Failure	MTTR (Mean Time to Recovery)	Measures how long the system remains in an unhealthy state before it returns to normal operation.	RecoveryTime − FaultInjectionTime
Post-Failure	MTTResolve (Mean Time to Resolve)	Represents the full incident lifecycle duration, from failure injection to final resolution (i.e., stable recovery).	ResolveTime − FaultInjectionTime

3.4 Experiment Procedure

The experiment compares the performance of a traditional monitoring setup with the proposed intelligent AIOps system. The baseline uses *Prometheus* for threshold-based alerts, representing conventional DevOps monitoring. In contrast, the ML-based system employs semantic embeddings and an autoencoder model for unsupervised anomaly detection. Both systems monitor the same *FastAPI* demo application, where controlled faults are injected to simulate host- and application-level incidents. Figure 5 illustrates the overall experimental architecture.

Fault injection covers multiple scenarios, such as CPU bursts, stopping services, and HTTP 5xx errors, to ensure the fairness of the experiment. For each experiment, a script records key timestamps marking fault injection, anomaly detection, repair, and resolution. These logs are then used to calculate three time-based metrics—**MTTD**, **MTTR**, and **MTTResolve**—providing a quantitative comparison of responsiveness and recovery efficiency between the two monitoring systems.

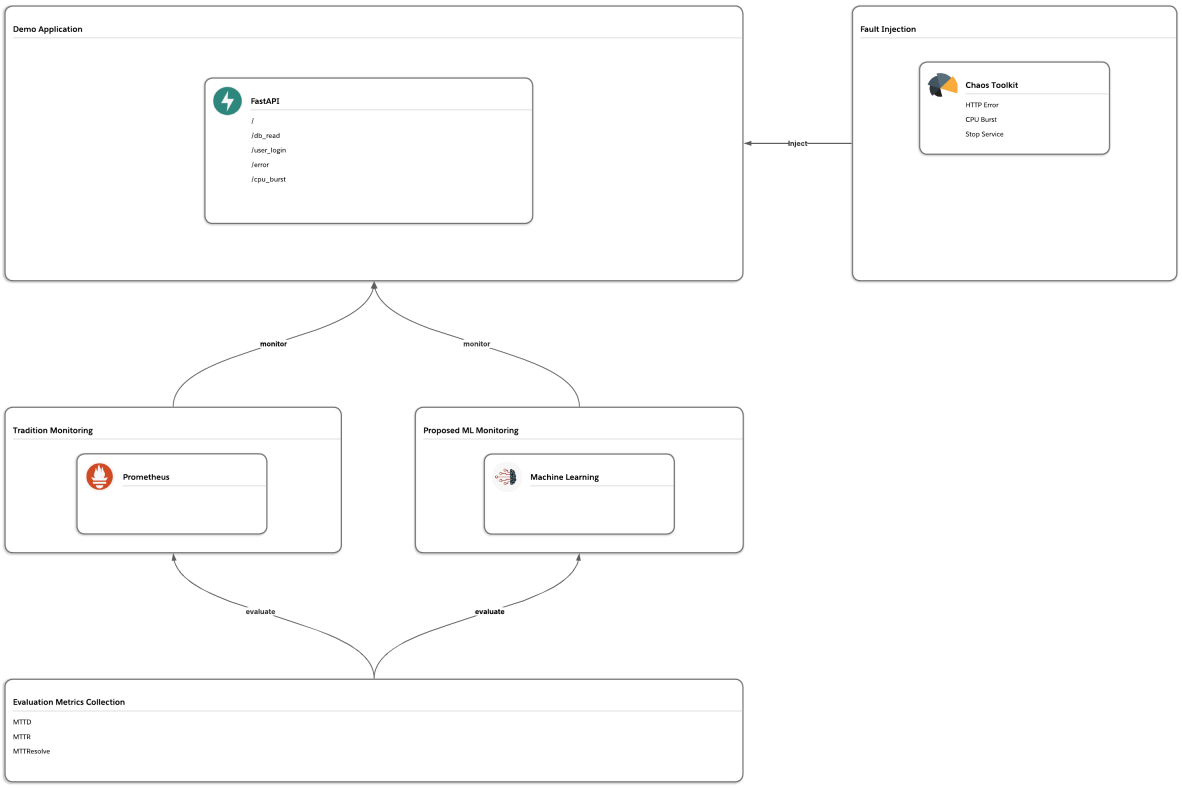

Figure 5: Experiment Architecture Design

4 Design Specification

4.1 Architecture

The architecture is organized into three layers—**Pre-Failure**, **In-Failure**, and **Post-Failure**—with a dedicated machine learning task supporting each stage. Each layer trains its own model using the preprocessing dataset, while the Post-Failure stage performs incremental training to incorporate newly validated incidents without retraining from scratch. Two real-time inference services execute the prediction and diagnosis models. Together, these components form a continuous closed-loop workflow that enables proactive detection, in-incident analysis, and post-incident learning. Figure 6 The Proposal Architecture Diagram.

4.2 Research Tool

Table 3 summarizes the tools used across different stages of the pipeline.

5 Implementation

5.1 Overview

This section describes the final implementation phase of the proposed AIOps monitoring framework, which follows the layer-based architecture designed earlier. The system

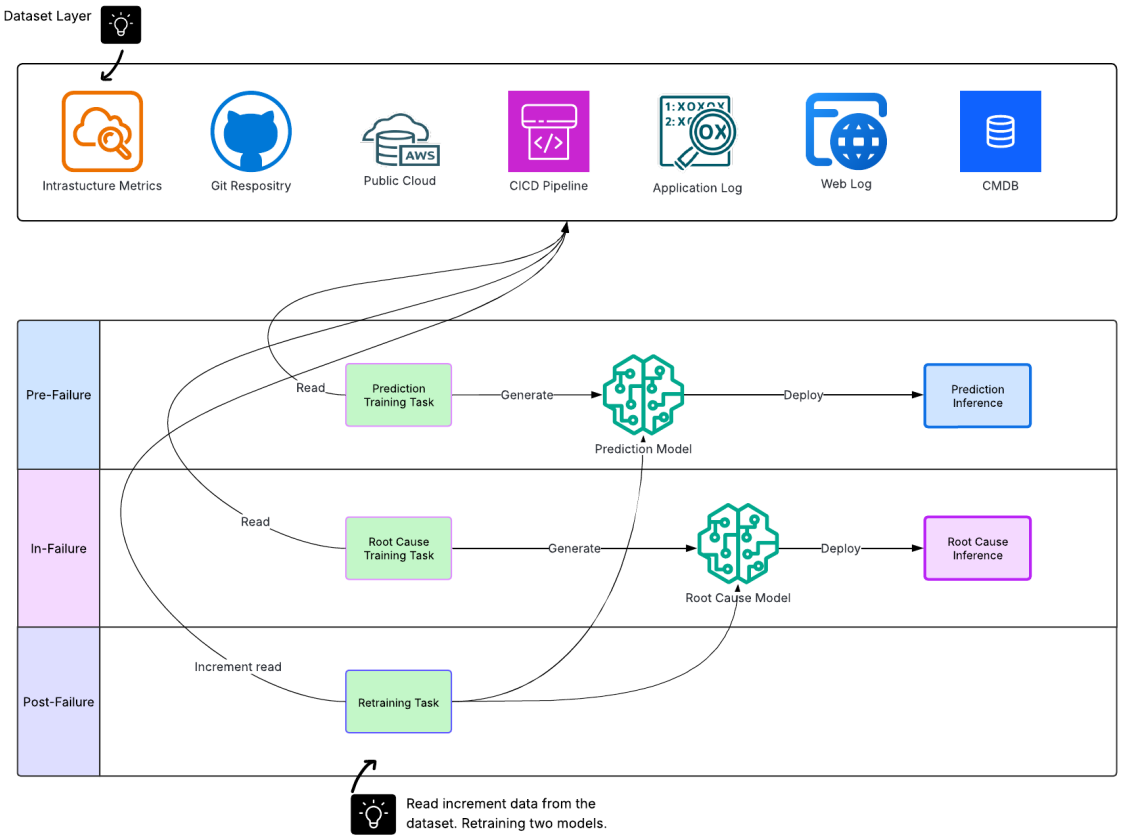

Figure 6: The Proposal Architecture Diagram

integrates data ingestion, messaging, processing, model training, and real-time inference into a unified monitoring pipeline.

5.2 Layer-based Implementation

As shown in Figure 7, the implementation follows a hierarchical structure, ensuring that each layer is modular yet connected across the data flow.

Data Collection Layer. Logs, metrics, and events were gathered from heterogeneous sources using *Vector*, a high-performance agent for unified log collection. All data were standardized into JSON format to ensure consistency and downstream compatibility. **Output:** structured JSON event stream.

Messaging Layer. *Kafka* served as the distributed broker between data producers and consumers, handling real-time message delivery between Vector and Spark. **Output:** Kafka topics with real-time event messages.

Data Preparation Layer. A *Spark Streaming* job consumed data from Kafka, parsed the JSON messages into semantic sentences, and transformed them into vectorized representations. **Output:** preprocessed embeddings in a Parquet file.

Model Development. Using the training datasets, an *Autoencoder-based* anomaly detection model and a root cause model were trained to learn normal operational patterns, along with the incremental retraining of the root cause model using labeled feedback samples. **Output:** trained prediction and root-cause models.

Model Deployment. The real-time inference jobs were deployed on Spark. As

Table 3: Summary of Research Tool

Tool	Used In Phase	Description	License
Vector	Data Collection	Log collector for multiple data sources	Open Source
Nginx	Data Collection	Web server providing access logs	Open Source
GitHub	Data Collection	Git commit and metadata logs	Free
GitHub Actions	Data Collection	CI/CD pipeline events	Free
Kafka	Data Processing	Distributed message queue	Open Source
Spark	Data Processing	Distributed data preprocessing engine	Open Source
Scikit-learn	Model Development	Python ML library for unsupervised models	Open Source
Sentence Transformers	Model Development	Semantic embedding generator	Open Source
FastAPI	Experiment	Demo backend and fault injection API	Open Source
Prometheus	Experiment	Baseline monitoring system	Open Source
Chaos Toolkit	Experiment	Automated fault injection tool	Open Source
Docker	Deployment	Containerization and orchestration	Open Source
AWS EC2	Deployment	Elastic compute infrastructure	Paid
Python	All Phases	Core programming language	Open Source

new events arrived from Kafka, each was vectorized and fed into the trained model for inference. Detected anomalies were logged automatically to a file. **Output:** continuous anomaly logs (`anomaly.jsonl`). And two real-time inference services.

In summary, this implementation achieves an end-to-end AIOps pipeline that automates data ingestion, model inference, and feedback collection—providing the operational foundation for evaluation in the next section.

6 Evaluation

6.1 Experiment / Case Study 1 - Traditional vs Machine Learning-based Monitoring

6.1.1 Experiment Description

To measure the performance of the two monitoring systems. We used a chaos engineering tool named **Chaos Toolkit** to simulate system faults (e.g., CPU bursts, HTTP errors, and Stop services) and automatically record all event timestamps. To ensure that the experiment can be repeated consistently, the chaos script was designed to perform the following sequence of actions: fault injection, execution of Prometheus and machine

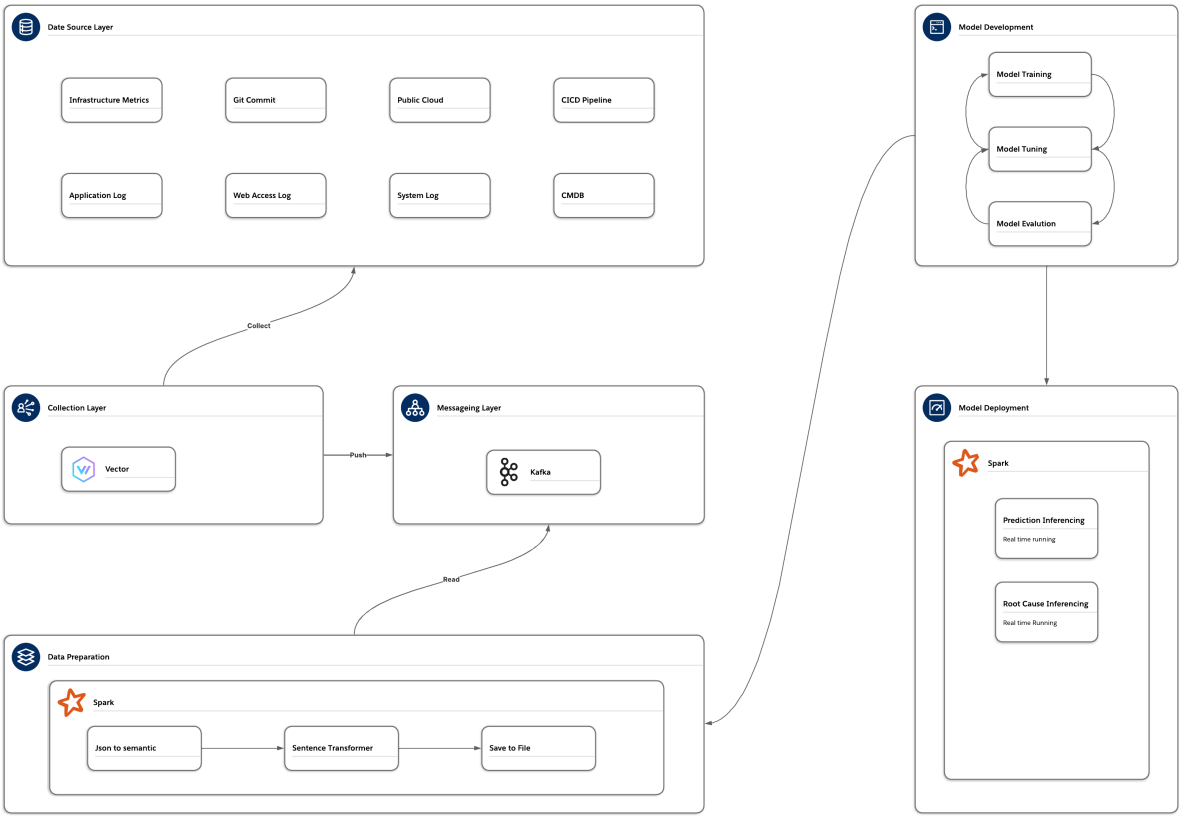

Figure 7: ML-Monitoring Architecture

learning-based monitoring detection scripts, execution of recovery scripts, and finally, automated calculation of key performance metrics. The metrics were derived by computing the time differences between critical events, such as **Detection Time Injection Time** and **Recovery Time Detection Time**, thereby obtaining comparative results for both monitoring systems.

6.1.2 Experiment Result

The image below compares the metrics of traditional and machine learning-based monitoring systems when monitoring a unified demo. Figure 8 Monitoring Performance Comparison (Traditional vs Machine Learning-based).

6.1.3 Discussion and Analysis

Prometheus's pull-based architecture relies on fixed scraping intervals and rule-evaluation windows, which inherently introduce detection latency. In contrast, the machine learning-based monitoring system performs continuous inference over streamed data, enabling near real-time anomaly detection.

To ensure a fair comparison, the chaos experiments were designed so that all injected faults could be detected by both monitoring approaches. Prometheus was explicitly configured with the necessary alert rules for each failure scenario, while the machine learning-based system operated without any predefined thresholds or alert policies.

Operationally, Prometheus requires engineers to manually define and maintain alert rules, thresholds, and durations. As the number of services or metric dimensions grows, so

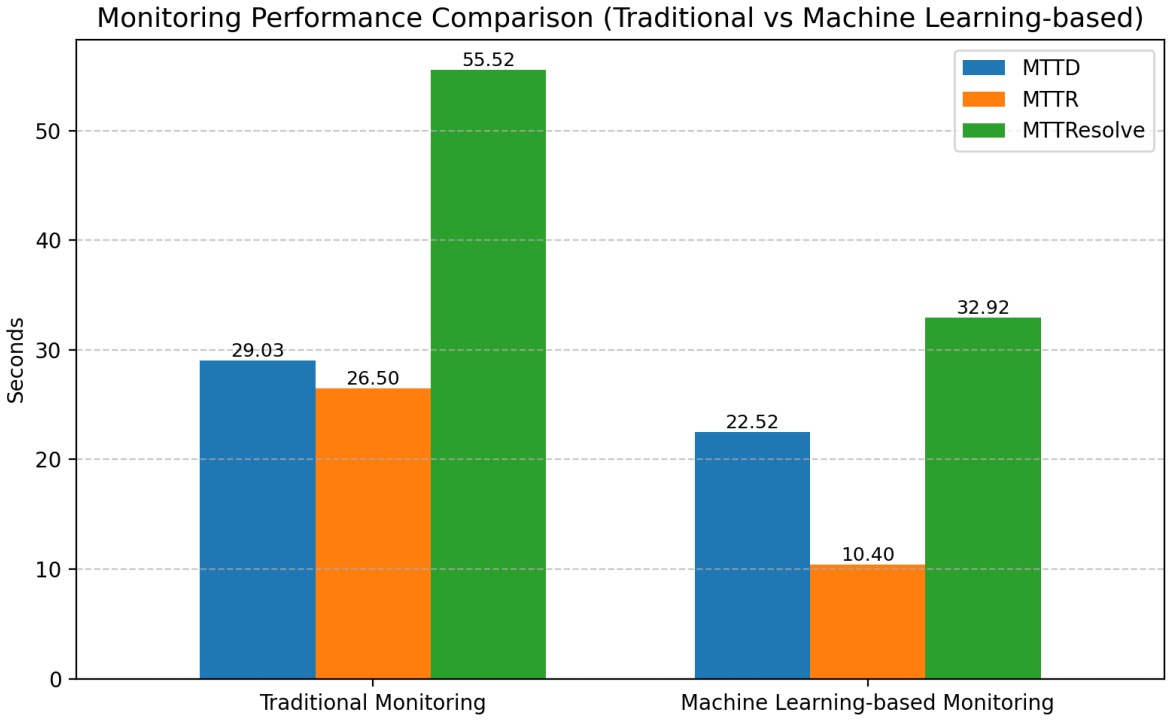

Figure 8: Monitoring Performance Comparison (Traditional vs Machine Learning-based)

does the volume of required alert configurations, increasing both complexity and maintenance overhead. More critically, rule-based systems depend on engineers' prior knowledge of possible failure modes. Any scenario not anticipated in the alert configuration may go undetected, representing a fundamental limitation.

Machine-learning-based monitoring avoids this constraint by learning normal operational patterns directly from data rather than relying on predefined rules. As a result, it can detect previously unseen anomalies and substantially reduce the risk of missed alerts—addressing one of the most persistent weaknesses of traditional threshold-based monitoring.

6.1.4 Summary

In summary, the machine learning-based monitoring framework outperformed the traditional monitoring system, **reducing MTTD by around 20%, MTTR by around 50% and MTTResolve by around 40%** under identical conditions. Although MTTResolve also decreased substantially, this metric mainly reflects the total experiment duration rather than an actual service restoration time, since the recovery phase was automatically terminated after anomaly detection. These results confirm the effectiveness of real-time, model-driven monitoring in achieving faster anomaly detection and recovery recognition.

6.2 Experiment / Case Study 2 - Structured Data vs Semantic Embedding

6.2.1 Experiment Description

In the initial phase, the training dataset was constructed using a JSON-based schema, where all key-value pairs were flattened into a wide structured table. This approach resulted in a dataset containing many fields, each requiring manual conversion into numeric or one-hot encoded values. Although conceptually clear, this structure led to a high degree of hard-coded feature engineering and poor model performance. Recognizing the scalability and maintainability limitations of this approach, we transitioned to a semantic representation method, where each data row was transformed into a natural-language sentence and encoded through a sentence transformer model (*all-MiniLM-L12-v2*). This allowed the model to infer the semantic meaning of each event—without depending on explicit field mappings or handcrafted rules. In this case study, we used approximately 5000 data points to train each model, with approximately 1000 rows that included 50 anomalies as a test set. We then conducted prediction tests and calculated their Precision, Recall, F1, and MSE to compare the two models.

6.2.2 Experiment Result

The image below shows a performance comparison of two models trained based on structured and semantic patterns. Figure 9 Semantic vs Structured Model Performance.

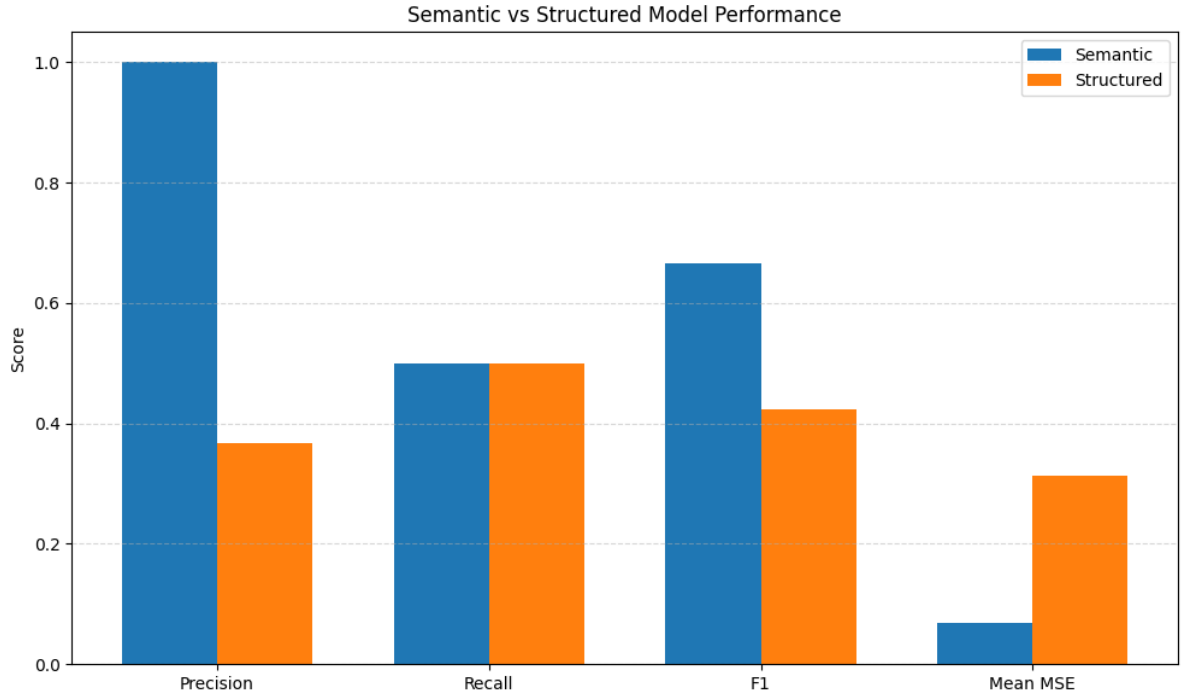

Figure 9: Semantic vs Structured Model Performance

6.2.3 Discussion and Analysis

Most importantly, collecting this type of monitoring data in real-world scenarios is a massive and complex undertaking. First, all fields need to be defined during preprocessing, and then standardized or one-hot encoded data needs to be applied during training. This presents the first challenge of manually maintaining such a large number of fields, which undoubtedly contradicts the automation of DevOps theory. Secondly, semantic models are trained to interpret anomalies in data, much like humans do, to identify problems. This is exactly what we hope for: instead of having to define alarms ourselves like traditional monitoring, we want the model to tell us what anomalies the system might be experiencing. Therefore, we have not explicitly configured any alerting strategies. This is the second breakthrough: semantic models can eliminate the static alerting strategies found in traditional monitoring systems.

On the other hand, the semantic model also significantly reduces alert noise. Instead of relying on rigid threshold rules—which often trigger alarms due to harmless metric fluctuations—the model evaluates log content itself to determine whether an event appears anomalous. This content-based judgement avoids many false positives generated by traditional numeric thresholds, addressing one of the long-standing limitations of threshold-based monitoring systems.

6.2.4 Summary

Overall, the results confirm that semantic embeddings provide a more expressive and robust foundation for anomaly detection. Compared with the structured wide-table approach, **the semantic model delivers higher accuracy, fewer false positives, and more stable reconstruction** behavior, while also simplifying preprocessing. These advantages make semantic embedding a more scalable and generalizable solution for machine learning-based monitoring.

6.3 Experiment / Case Study 3 - Root Cause Analysis and Feedback Learning

6.3.1 Experiment Description

During the troubleshooting phase, identifying the faulty service is often one of the most time-consuming tasks for DevOps engineers. In the post-incident phase, failure reviews are fully manual, which limits reuse of historical knowledge. To address this problem, our system performs real-time root cause analysis and utilizes manually confirmed failure events as feedback data for incremental learning.

In this experiment, anomalies detected by the semantic model are manually confirmed, simulating the real-world oncall workflow. Once an anomaly is validated, the root cause analysis model generates a ranked list of candidate services by combining semantic similarity with CMDB information. After the incident is resolved, the confirmed data is added to the feedback dataset, and the root cause analysis model is incrementally retrained. This allows us to evaluate whether feedback learning improves the model's ability to pinpoint failures.

6.3.2 Experiment Result

The table below shows the changes in confidence levels of the root cause analysis model after training on CMDB data alone and after training on the feedback dataset.

Table 4: Confidence Gain Before and After Feedback Learning

Log Type	Service	Baseline	Feedback	Gain
Nginx Error	nginx	0.465	0.743	+59.8%
App Log	fastapi-demo	0.368	0.825	+124.1%
Stop Service	fastapi-demo	0.381	0.850	+123.3%
Average	–	**0.417**	**0.832**	**+101.5%**

6.3.3 Discussion and Analysis

In a real enterprise environment, the CMDB records information about all services and middleware. Therefore, if a problem occurs, it must be a problem with one of these services. Fault localization is essentially about locating a specific service within the CMDB. During the in-progress phase, the most important thing is how to quickly identify problematic applications. Due to the complexity of the actual environment, it may be difficult to quickly locate them. The semantic associations of the root cause analysis model combined with the global perspective of the CMDB can help us accelerate problem analysis.

Recording and reviewing past problems is an essential part of production, but relying solely on manual organization is inefficient. We improve the accuracy of root cause analysis by retraining the model based on faults identified in the production environment. Root cause analysis models and feedback learning greatly improve the parts of traditional monitoring processes that are not automated. This is precisely why everyone continues to explore machine learning in various industries: to enable machines to do things that humans are better at.

6.3.4 Summary

By integrating machine learning into both the in-failure and post-failure phases, the system improves the efficiency of fault identification. The experiments show that the root cause model can correctly associate anomalies with CMDB entities, and feedback learning further enhances this ability—**the confidence has greatly improved.** This demonstrates that incorporating confirmed incidents enables the root cause analysis model to continuously refine its accuracy.

6.4 Discussion

6.4.1 Interpretation of Findings

After studying three case studies and implementing an entire intelligent monitoring system, we concluded that integrating machine learning into monitoring systems can significantly improve the MTTD, MTTR, and MTTResolve functions of application systems. Throughout the entire event lifecycle, efficiency is significantly higher than that of traditional monitoring systems. Furthermore, it reduces configuration items and integrates multiple monitoring metrics, achieving a higher level of monitoring dimensionality.

6.4.2 Critical Assessment

First, the monitored application was too small, as it was a single-point backend application. In real-world environments, the monitored application might be much larger, not just a single service. It might consist of a series of components, including front-end, back-end, middleware, and databases, which make monitoring more complex. Therefore, a better approach would be to deploy the experiment within an enterprise.

Second, regarding machine resources, due to cost constraints, We only purchased low-spec machines for the experiment, which led to resource usage issues. The experimental components themselves also consumed a lot of resources. A more rigorous approach would be to separate the monitored system from the monitoring system.

Third, regarding data sources, we all know that the most critical factor in the quality of machine learning models is the quality of the data source. While we did our best to deploy all the necessary components found in real-world environments, such as Nginx and CICD pipelines, and simulated real logs, in reality, as mentioned earlier, the complexity and variety of components are much greater. The data sources for intelligent monitoring systems should be as comprehensive as possible. The data we simulated is still quite limited.

7 Conclusion and Future Work

7.1 Answers to the Research Questions

This section revisits the main research question and its five sub-objectives defined previously, summarizing how each objective has been achieved.

Main Research Question

> *How much MTTR can be reduced by a machine learning-based intelligent monitoring system compared to traditional monitoring systems in DevOps environments?*

Answer

> The proposed machine learning-based monitoring system reduced **MTTR by approximately 50 %** compared with traditional monitoring, confirming that machine learning integration enables faster fault detection, diagnosis, and recovery throughout the DevOps lifecycle.

Sub-objectives

1. **Data Source Identification:** Case Study 1 shows that diverse observable data—such as logs, metrics, and application events—provide the most relevant inputs for improving anomaly prediction.

2. **Data Preprocessing:** Case Study 2 shows that a semantic-embedding–based preprocessing strategy yields more accurate and efficient anomaly detection compared with traditional wide-table feature engineering.

3. **Deployment Integration:** The Vector–Kafka–Spark Streaming pipeline can be integrated as a non-intrusive streaming layer, enabling real-time model serving without modifying existing DevOps workflows.

4. **Evaluation Results:** Case Study 1 confirms that time-based metrics such as **MTTD**, **MTTR**, and **MTTResolve** provide an effective and reliable way to evaluate monitoring system performance.

7.2 Conclusion

This research examined whether a machine learning–based monitoring framework could enhance system reliability compared with traditional threshold-based monitoring. A complete intelligent monitoring pipeline was implemented, covering data acquisition, preprocessing, model development, and deployment. Using a chaos-driven experimental setup enabled a reproducible, quantitative comparison between Prometheus and the proposed machine learning-based system. Results show that machine learning can substantially reduce detection and recovery latency (MTTD, MTTR, and MTTResolve), thereby improving system availability. These findings bridge the gap between theoretical AIOps research and practical implementation, demonstrating measurable improvements in reliability through machine learning-based monitoring.

7.3 Future Work

Future work should address the current limitation of lacking time-window–based prediction. Certain anomalies emerge only through temporal patterns rather than single-point deviations, and traditional monitoring tools often support this capability. Incorporating temporal modeling would therefore strengthen the system rather than replace existing advantages. In addition, we envision broader community involvement in extending this framework. Contributions from DevOps practitioners and researchers could help evolve it into a mature open-source platform and a representative solution for intelligent monitoring in production environments.

References

Abbas, S. I. and Garg, A. (2024). Aiops in devops: Leveraging artificial intelligence for operations and monitoring, *2024 3rd International Conference on Sentiment Analysis and Deep Learning (ICSADL)*, pp. 64–70. https://doi.org/10.1109/ICSADL61749.2024.00016 Cited by 7.

Alenezi, M., Zarour, M. and Akour, M. (2022). Can artificial intelligence transform devops?, *ArXiv* **abs/2206.00225**. https://arxiv.org/abs/2206.00225 Cited by 12.

Ali, M. S. and Puri, D. (2024). Optimizing devops methodologies with the integration of artificial intelligence, *2024 3rd International Conference for Innovation in Technology (INOCON)*, pp. 1–5. https://doi.org/10.1109/INOCON60754.2024.10511490 Cited by 18.

Beyer, B., Jones, C., Petoff, J. and Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems*, 1st edn, O'Reilly Media, Inc.

Dang, Y., Lin, Q. and Huang, P. (2019). Aiops: Real-world challenges and research innovations, *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 4–5. https://doi.org/10.1109/ICSE-Companion.2019.00023 Cited by 222.

Hany Fawzy, A., Wassif, K. and Moussa, H. (2023). Framework for automatic detection of anomalies in devops, *Journal of King Saud University - Computer and Information Sciences* **35**(3): 8–19. https://doi.org/10.1016/j.jksuci.2023.02.010 Cited by 17.

Jeyarajan, B., Murugan, A., Pandy, G. and Pugazhenthi, V. J. (2025). Ai for predictive monitoring and anomaly detection in devops environments, *SoutheastCon 2025*, pp. 450–455. https://doi.org/10.1109/SoutheastCon56624.2025.10971552 No cited.

Shen, S., Zhang, J., Huang, D. and Xiao, J. (2020). Evolving from traditional systems to aiops: Design, implementation and measurements, *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*, pp. 276–280. https://doi.org/10.1109/AEECA49918.2020.9213650 Cited by 22.

Shrivastava, S., Srivastav, N., Artasanchez, A., Sayed, I. and Ph.D, D. S. C. (2023). *AWS for Solutions Architects: The definitive guide to AWS Solutions Architecture for migrating to, building, scaling, and succeeding in the cloud*, Packt Publishing.

Soldani, J. and Brogi, A. (2022). Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey, *ACM Comput. Surv.* **55**(3). https://doi.org/10.1145/3501297 Cited by 222.

Teggi, P. P., N., H. and Malakreddy, B. (2022). Aiops based predictive alerting for system stability in it environment, *2022 International Conference on Innovative Trends in Information Technology (ICITIIT)*, pp. 1–7. https://doi.org/10.1109/ICITIIT54346.2022.9744236 Cited by 5.

Thantharate, P. (2023). Intelligentmonitor: Empowering devops environments with advanced monitoring and observability, *2023 International Conference on Information Technology (ICIT)*, pp. 800–805. https://doi.org/10.1109/ICIT58056.2023.10226123 Rank C.

Valli, L. N., Sujatha, N. and Rathinam, E. J. (2023). A study on deep learning frameworks to understand the real time fault detection and diagnosis in it operations with aiops, *2023 International Conference on Evolutionary Algorithms and Soft Computing Techniques (EASCT)*, pp. 1–6. https://doi.org/10.1109/EASCT59475.2023.10392382 Cited by 6.