National College of Ireland

Machine Learning based Ensemble Anomaly Detection in Kubernetes Environments

MSc Research Project
Cloud Computing

Rohit Korlahalli
Student ID: 23303395

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rohit Korlahalli
Student ID:	23303395
Programme:	M.Sc in Cloud Computing
Year:	2025-26
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	Machine Learning based Ensemble Anomaly Detection in Kubernetes Environments
Word Count:	6456
Page Count:	22

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:	Rohit Korlahalli
Date:	11/12/2025

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):	

Machine Learning based Ensemble Anomaly Detection in Kubernetes Environments

Rohit Korlahalli
23303395

Abstract

The emergence of containerised workloads and the concept of microservice have changed application deployment but has equally complicated the task of monitoring because of dynamic and distributed nature of Kubernetes environments. The traditional threshold-based monitoring tools are generally not able to give proactive scaleable or interpretable information about system behavior. The presented research is a lightweight ensemble machine learning model that is intended to identify an anomaly in real-time through Kubernetes telemetry. This system is a combination of Isolation Forest, XGBoost, and Long Short-Term Memory (LSTM) models which model distributional, structural and temporal patterns in Prometheus metrics. Such individual predictions are combined by using an ensemble of majority votes to become more robust to a variety of anomalies. The ensemble inference service which is deployed on Google Kubernetes Engine (GKE) with FastAPI handles live metrics and makes its resulting publications again through Pushgateway to Prometheus. Grafana dashboards visualize trends of anomalies and the health of the system, and a natural-language assistant (not compulsory but may be added) can give natural-language explanations and recommendations on how systems should work. Notifications and insights are also sent to Slack to assist with intervening before it is too late. Analysis based on real and synthetic workloads shows that the ensemble is superior to single models with regard to stability and the general reliability of detection, overcoming the most critical limitations revealed in the literature. The results prove that the use of lightweight ML models together with existing observability pipelines could be used to deliver accurate, interpretable, and operationally viable anomaly detection to cloud-native environments.

Keywords: Kubernetes, Anomaly Detection, Ensemble Learning, Isolation Forest, XGBoost, LSTM, Prometheus, Grafana, Google Cloud, Cloud Native.

1 Introduction

Kubernetes is a container orchestration platform that has entirely transformed the manner in which workloads are rolled out and controlled, particularly in distributed microservices based structures. Its scaling capabilities and automation of processes make it one of the platforms that the organizations can count on to deploy quickly across clusters. These capabilities of Kubernetes however have made observability more complex. The traditional monitoring systems, compare workload metrics with pre-determined thresholds, this is the reason most of the times the traditional monitoring systems cannot

account non linear, variable behaviour of workloads on Kubernetes clusters.

Various reasons may lead to anomalies in distributed systems including resource saturation, resource configuration errors, transient spikes, or faulty deployments that can rapidly propagate to other dependent services in a microservices based system. Such workloads are therefore much required to be actively monitored with the aim of achieving increased system availability and uptime. The implementation of anomaly detection with the help of the machine learning (ML) or artificial intelligence (AI) can prove to be highly effective in assuring the presence of an active based monitoring strategy. The recent literature on AI in IT operations (AIOps) highlights the application of Machine Learning (ML) models to detect abnormal behavior in a system, mitigate alert fatigue, and make predictive maintenance.

This research addresses this challenge by proposing a Machine Learning-based Ensemble Anomaly Detection Framework specifically for Kubernetes workloads. The framework uses a combination of Isolation Forest, XGBoost, and LSTM models to predict anomalies in container telemetry data, captured through Prometheus. The ensemble mechanism enhances detection robustness by combining tree-based, gradient-boosted, and time-series models.

The framework does not only present numeric scores of anomalies, but it also presents human readable insights in the form of being interpretable to the DevOps teams. The visualization with real-time alerting is made available by the integration with Pushgateway, Grafana, and Slack. Such insights as High CPU Utilization - consider adjusting HPA tresholds are used to allow the engineer to act immediately, like Memory nearing saturation - investigate pod behavior.

The primary objectives of this research is to determine:

To what extent can an ensemble-based machine learning framework integrated with Kubernetes observability pipelines accurately detect anomalies, remain interpretable for DevOps teams, and provide actionable, explainable insights that support operational decision-making in cloud-native environments?

The project adds to the current research on AIOps because it shows how ensemble learning can be used to achieve better observability, fault tolerance, and proactive system governance of cloud-native setups.

2 Related Work

In this section, the critical review of existing academic literature concerning machine learning-based anomaly detection and observability in Kubernetes environments is presented. It is intended to assess what other researchers have already accomplished, recognizing the advantages and shortcomings of their methods, and how these studies can enlighten, contradict, or otherwise fail to satisfy the requirements of this study. This review presents the background of the research and offers a clear rationale of the research question and the

ensemble-based approach taken in this project by examining the strengths and limitations of the previous studies.

2.1 Machine Learning-Based Anomaly Detection in Microservices

The strong growth seen in the adoption of cloud-native systems and applications has seen the encouragement of applying Machine Learning (ML) techniques to detect behavioural anomalies in Kubernetes and microservices setup. One of the most interesting works in this area is the Kubernetes Anomaly Detector (KAD) presented by Konsinska and Tobiasz (2022) . Their study introduced an adaptive hybrid systems using Long Short-Term Memory (LSTM) networks, Autoencoders and SARIMA models for analyzing Prometheus like telemetry. Their contribution stresses on the use of time series learning while looking for anomalies in the microservices architecture as they have the capabilities to find subtle deviation within the Kubernetes workloads. While the model selection makes the model more robust, KAD is computationally intensive and does not provide much information on the feasibility of deployment in real time or integration into the existing observability tools. Their work also does not offer any actionable operational interpretation that can help the Devops team to work on the alerts effectively.

A related study by De Silva et al. (2022) used kernel metrics derived using eBPF with Autoencoders and LSTMs to identify microservice anomalies. Although fine grained metrics help in achieving strong detection accuracy, it depends on privileged kernel access , making it unsuitable for cloud managed offering like Google Kubernetes Engine (GKE) , AWS Elastic Kubernetes Service (EKS) etc. Like KAD this study also highlights the benefits of using LSTM based temporal model but is limited by the use of a single model design and lack of an ensemble or multi model strategy.

Some of the recent works have brought anomaly detection more towards the aspect of proactive monitoring. Podduturi (2025) showed that Autoencoders, LSTMs and Reinforcement learning can be better than static thresholds based systems capturing complex behavioral deviations. Similarly, Almaraz-Rivera (2023) created a multi-model pipeline based on Isolation Forest, One-Class SVM and Autoencoders that was validated with chaos-engineering workloads. Although these sorts of studies show the value of richer ML techniques, they don't integrate with Prometheus or Grafana and don't deal with alert noise or challenges of interpretation.

Research has also moved towards explainability and efficiency. Shi et al. (2023) introduced Alioth, and explainable anomaly detection system using Denoising Autoencoders and transfer learning, and although it depends on VM-level hardware, it is not relevant to the pod-level Kubernetes workloads. Complementing this, Zhang et al. (2025) demonstrated that even using lightweight ML models can be an effective way to detect anomalies in resource constrained edge Kubernetes environments, again stressing on the need for efficient and deployable solutions.

Overall, literature exhibits that models like LSTMs, Isolation Forest, and Autoencoders are effective in detecting the temporal and structural anomalies for the Kubernetes and microservice workloads. However , prior works rarely combine different models prediction into an ensemble strategy , integrate ML predictions directly into Prometh-

eus observability pipelines, or provide clear, interpretable insights insights for operations teams. These gaps together serve as a driving force for the design and contribution of the current research.

2.2 Real-Time Observability and Monitoring in Kubernetes

Much of the literature regarding Kubernetes mainly focuses on observability pipeline than ML driven interface. In their paper Hasan and Abdullah (2022)presented Kube-Monitor, a system to gather pod and node metrics using Prometheus, and visualize them in Grafana.Though its effective for real time visibility it is reactive in its approach and is dependent on static thresholds. Similarly ,Henao Villa et al. (2025) demonstrated the use of Prometheus and synthetic workloads for automated performance monitoring , confirming Prometheus as a reliable telemetry source but once again it lacked proactive monitoring and anomaly detection. More advanced things, like the study by IEEE (2024) on the security of Kubernetes using ML (Machine Learning) make evident that there is potential for AI to improve observability, although it covers areas of concern for security rather than operational performance which is why they do not incorporate observability pipelines. Overall, existing observability systems rely mainly just on visualizations, static thresholds and do not really looking into proactive monitoring and using ML for monitoring and reducing alert noise and improving alert insights for effective mitigation. This creates an important requirement for better integration of ML inference with Kubernetes native stack monitoring.

2.3 Research Gap

The literature, time after time, has uncovered three main gaps: the need for a ML-based proactive anomaly detection pipeline built directly on Prometheus telemetry, ensemble methods empowering the mix of temporal, unsupervised and tree-based models, and the lack of availability of interpretable and therefore actionable insights for the DevOps teams. No prior work has unified these elements inside of an operationally deployable system. This study addresses these gaps by creating an ensemble-based anomaly detection API coupled with Prometheus and augmenting it with LLM-powered insights to support real-time decision-making.

2.4 Summary of Related Work

From the advent of ML and deep learning algorithms, the literature tells us that ML algorithms including LSTMs, Autoencoders and Isolation Forests have been a success when it comes to finding anomalies in Kubernetes and microservice workloads, and that Prometheus and Grafana are the go to for cluster observability. However, currently existing approaches also have the shortcomings of being limited to single-model designs, reactive monitoring, weak integration between ML inference and observability pipelines, and lack of actionable or interpretable insights. These gaps underscore the importance of having a unified, lightweight ensemble API integrated with Prometheus and assisted by the guidance of LLM-driven recommendations which this research will address.

3 Methodology

3.1 Research Approach and Rationale

This research experimental, data-driven approach was based on the methods that were already applied in Kubernetes anomaly detection studies. A quantitative method was chosen for the performance measure of the three ML models - Isolation Forest, XGBoost and LSTM - and their combined ensemble to be measured objectively. This approach enables the study to be built on top of techniques used in previous research, but also to extend them through a uniform pipeline that can be deployed in Kubernetes and integrated with Prometheus for real-time monitoring.

3.2 System Setup and Data Collection

All of the experiments were carried out on Google Kubernetes Engine (GKE) cluster with Prometheus used to scrape telemetry. A test microservices application was set-up that was forked from a public git repository, this ensured real practical metrics are fetched and the training data depicts real production environment. Metrics such as CPU, memory, network throughput, filesystem I/O and pod restarts were collected at 15 seconds interval. To record normal and abnormal behavior, synthetic stress (CPU, memory, and network load) were introduced as well as regular workloads. Telemetry was exported as CSV files which was used as a dataset for model training and testing.

3.3 Data Preprocessing

The raw telemetry was structured into a time-ordered dataset with feature vectors aligned to timestamp. Missing values were dealt with, metrics were normalized and labelled according to workload behavior - normal as 0 and anomaly as 1. The resulting dataset was imbalanced which is reflective of real world conditions and was taken care of later during the training of model through the use of weighting.

3.4 Model Development and Training

Three models were developed to give insights into differnet aspects of Kubernetes telemetry - Isolation Forest for unsupervised outlier detection, XGBoost healthy classification under class imbalance and LSTM for learning temporal patterns to detect subtle deviations. Training of the models were done on AWS Sagemaker Notebook instance for consistency , records and reproducibility.The data set was divided into training and testing data sets using stratified sampling and the models made predictions independently. These outputs were then combined using majority voting to create the ensemble prediction, to overcome limitations present in previous single model approaches, and make the predictions more robust to different types of anomalies.

3.5 Evaluation Methodology

Model performance was assessed based on accuracy, precision, recall and macro-F1 to handle the class imbalance. Confusion matrices were created to analyze true and false predictions. The ensemble was then compared directly with the individual models to determine if the combined ensemble models were more robust and had less misclassification.

3.6 Deployment and Real-Time Inference

The ensemble model was deployed as a FastAPI service on the cluster running on GKE and it has facilitated real time anomaly inference. Predictions of the API were pushed to Prometheus using Pushgateway as a custom metrics and visualised in Grafana dashboards. A LLM based Devops assistant (Ollama) was packaged along to provide a natural language interpretation of the anomalies assisting the teams to effective mitigation strategy.

4 Design Specification

4.1 System Architecture

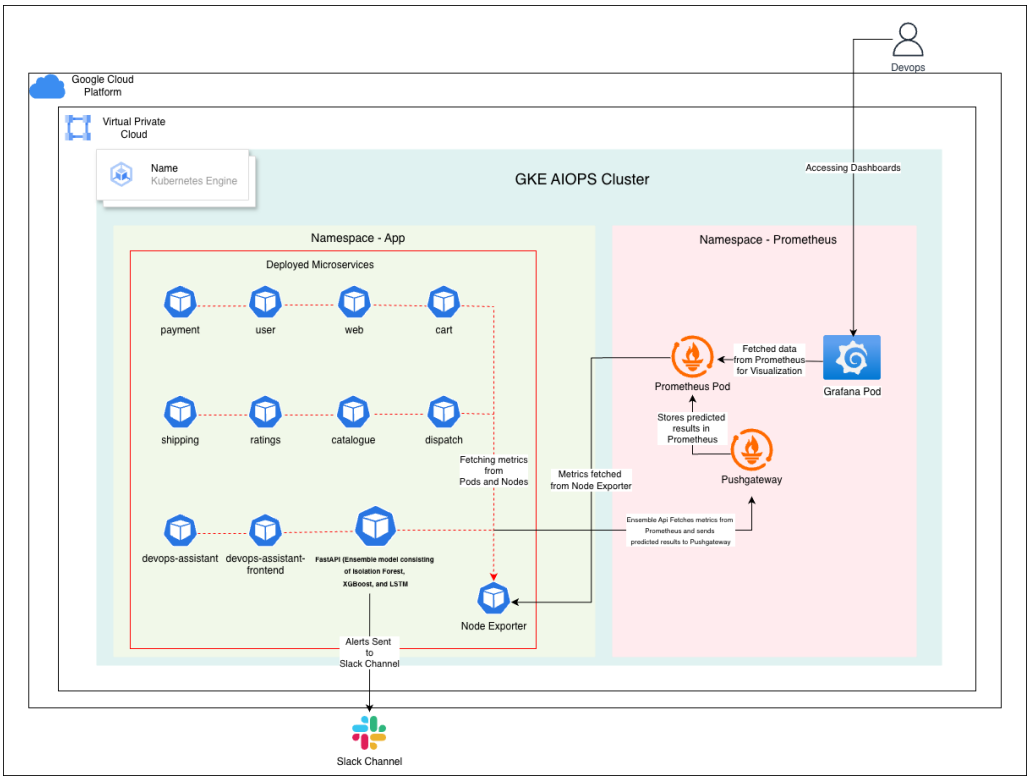

Figure 1: Architecture Diagram

The proposed system is designed as a comprehensive intelligent anomaly detection framework that coordinates very well with the Kubernetes native observability stack. It is created by integration of the following core components- Prometheus, the Ensemble Anomaly Detection API, Pushgateway, Grafana, and Slack - which together enable proactive monitoring, Machine learning based inference on real-time data and alerting. The system is designed in such a way that it can be easily scaled, interpreted, and deployed across cloud-native environments.

Prometheus, is used as the data collection layer, that is responsible for collecting all of the telemetry data for the deployed pods and containers on the Google Kubernetes Engine (GKE) cluster. It regularly scrapes the latest values for metrics like CPU Utilization, memory consumption, network throughput, and filesystem I/O activity, it stores

them as time-series data. These metrics are the base that are used for subsequent model training, anomaly detection and analysis. The Ensemble Anomaly Detection API, which is developed using FastAPI and python is the main analytical aspect of the framework. The Ensemble API houses the ML models - Isolation Forest, XGBoost, and LSTM. Every new metric is evaluated against all of the 3 models and the effective result is then shared , making the evaluation more robust and trustworthy. Once the predictions are made , basis the results, few insights are also shared for the Devops/Cloud teams for them to better decide on the course of action for a particular incident.

The output of the ML model inference along with the insights shared are the sent to Prometheus via Pushgateway, which acts as the temporary storage of the custom metrics generated by ML models. This makes sure that all of the relevant metrics are push backed to Prometheus which is the single source of truth and which stores the metrics in time-series format. This data helps to easily visualize in dashboards using Grafana along with standard metrics of pods and nodes. Grafana provides the ability to create intuitive dashboards, that can visualize anomaly trends, pod-level insights, heatmaps, and overall cluster health, which aid the SRE/Devops team to understand system health in a better way.

The last part of the framework is the Slack that receives the alerts from the Ensemble API along with insights based on the output of the ML inference i.e whether the latest metrics fetched is anomalous or not. The ensemble API also has a cool down mechanism for low-priority findings to prevent unnecessary noise in the slack channel. Combining ML inference with standard Kuberenets observability tools, makes sure that the system operates transparently with existing workflows and does not disturb existing monitoring practices. Overall, the architecture ensures a scalable AIOps framework that also keeps human intelligence and decision making in the loop while proactively monitoring distributed microservices based workloads on Kubernetes cluster.

4.2 Model Architecture

The model architecture consists of three complementary machine learning models, namely, Isolation Forest, XGBoost and LSTM, which are chosen to represent different behavioral dimensions of Kubernetes telemetry. These models collectively create the analytic core of the ensemble anomaly detection scheme.The preprocessing phase converts raw metrics (CPU, memory, network I/O, filesystem I/O, and container restarts) into a feature vector that can be input to all three models. This guarantees compatibility of features with models and uniform behavior among inference pipelines.

Isolation Forest Architecture:

Isolation Forest is an unsupervised feature space recursive partitioning based anomaly detector. It trains on mostly normal data and identifies the outlier samples with the aid of shorter tree paths. In the framework, it gives a stable baseline signal through detecting global outliers and distribution shifts. It is lightweight and therefore it enables high-frequency scoring in the real-time inference API.

XGBoost Architecture:

XGBoost serves as the supervised component of the architecture. It is based on the gradient-boosted decision trees to acquire more complicated decision-boundaries between typical and unusual telemetry patterns. The imbalance of classes is mitigated through the scale+weight parameter whereby the model is sensitive to the anomaly of minorities. The XGBoost part serves a powerful recall of sudden behavioral aberrations like CPU spikes or sudden network surges.

LSTM Architecture:

LSTM model learns time dependency of the telemetry stream. It is composed of multiple layers of LSTM and then a dense output layer that distinguishes each timestep as normal or anomalous. This model is the one that identifies gradual drifts e.g. memory leaks, slow resource depletion and changing workload patterns. The sequential behavior examination is its strength that complements the XGBoost short-term decision boundaries and Isolation Forest distribution-based approach.

Ensemble Architecture:

Majority-voting strategy combines the three models in the ensemble mechanism. All the models make binary predictions and the ensemble makes the final label of anomaly by agreement of at least two models. The design combines the advantages of each approach, which includes the stability of Isolation Forest, the sensitivity of XGBoost, and the temporal awareness of LSTM, and addresses the weaknesses of each. Its architecture is made deliberately lightweight to achieve a minimum of latency in the real-time inference in the FastAPI service.

The model architecture, in general, is specifically designed to be hybrid, integrating statistical and supervised and temporal learning methods into a single pipeline. With this design, the framework is resistant to various forms of anomaly and workload distributions in cloud-native environments. This is what gives the output a trustworthy and interpretable anomaly detection engine that can be suitably adapted to operational needs of Kubernetes-based microservices.

4.3 Ensemble Design Logic

The ensemble decision logic is the main part of the anomaly classification mechanism and is responsible for combining the outputs of the three independent models into a single reliable prediction. The ensemble is intended to take advantage of the complementary strengths of Isolation Forest, XGBoost and LSTM while reducing their weaknesses. This hybrid approach makes more stable detection of the diverse and often unpredictable behavior of Kubernetes workloads possible.

Each telemetry data point is evaluated across all of the three models and the result of all 3 models are taken into consideration for the ensemble logic. Isolation Forest adds the distribution based anomaly score, XGBoost adds the supervised classification approach based on the historical data patterns and the LSTM model adds the temporal view as it understands the sequential dependencies of the data. Each model makes a binary

determination by producing a binary output i.e normal (0) or anomaly (1). These predictions are then aggregated using a majority-voting strategy, which determines that an anomaly has occurred if two or more out of three models will determine the sample to be anomalous. This simple, but effective rule helps to prevent one model from overpowering the decision process and helps to reduce false positives that can occur with XGBoost due to its sensitivity, or false negatives that may occur with Isolation Forest.

The ensemble model doesn't just predict the anomaly , it also shares an insight based on the anomaly which is also shared to the team on slack channel, and stored in Prometheus and visualized in the Grafana dashboard. These include such indicators as which models caused the anomaly to occur, which is likely to have caused the deviation, and whether the anomaly looks like a sudden spike, a gradual drift, or a noisy behavioral change. By taking into consideration , the robustness of the ensemble model, the insights sharing and the easy deployment , this makes it very ideal to be implemented in the real-time Kubernetes environment for live API inference.

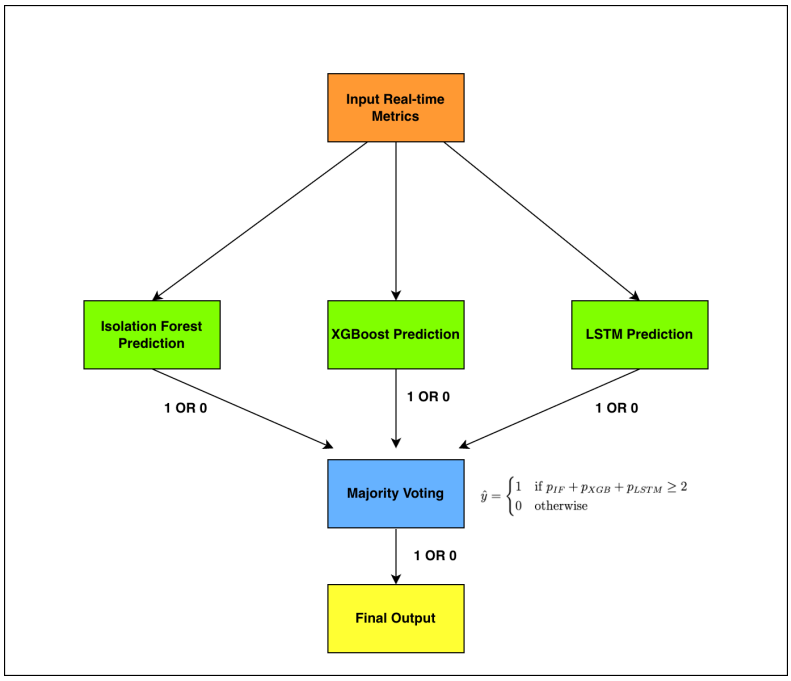

Figure 2: Ensemble Design Logic.

5 Implementation

This section is primarily focused on the final implementation of the proposed anomaly detection framework, and present the most important components developed, the tools that were used and the outputs that were produced. Here the focus is on the final solution that was implemented and not the intermediate versions of the framework that were tested, also there is no code or configuration part discussed here. The implementation brings together different segments like data processing, machine learning modeling, ensemble inference, deployment on Kubernetes and real-time monitoring.

The first step of the implementation was to prepare the telemetry data for workloads on Kubernetes by deploying a sample microservices setup with multiple databases and backend frameworks , that generate various system metrics , which makes the data more diverse and suitable for training the model. The microservices application that was deployed has been forked from a public github repository (Veeramalla (2021)) [1]. The microservices application are deployed on Google Cloud on the GKE (Google Kubernetes Engine) cluster. Raw metrics such as CPU utilization, memory consumption, network traffic, filesystem activity, and container restarts were collected and combined into a single structure. The data was cleaned , treated for missing values, normalised and exported as final preprocessed dataset (preprocessed\dataset\0510.csv). This dataset was used as the foundation for all training and testing of the model.

This dataset was used for training and development of three machine learning algorithms: Isolation Forest, XGBoost, and Long Short-Term Memory (LSTM). The 3 models were chosen to analyze different aspects of anomalous behavior: Isolation Forest is used for unsupervised outlier detection, XGBoost for supervised classification and LSTM for temporal sequence learning. Model was developed using Python using Scikit-learn, XGBoost, TensorFlow, Numpy and Pandas with training and hyperparameter evaluation done on Amazon Sagemaker. The final artefacts including the trained model files (iforest_model.pkl, xgboost_model.pkl, lstm_model.h5) and supporting scaler were exported for the integration in deployment pipeline. Outputs from this stage included classification reports, confusion matrix, prediction logs, graphs and saved model artefacts.

For more robustness and reliability, an ensemble mechanism has been implemented. This component takes evaluations of all three models for each telemetry data and based on the opinion of all the three models categorizing the metric as anomalous or not a decision based on the majority of votes is taken. The primary reason for choosing three models is to ensure we have a result and no split brain decisions which could make decision making impossible. The main advantage of the ensemble strategy is that it complements the strengths of all of the three models, and the decisions are more stable and reliable. The evaluation of the ensemble logic has been implemented in Sagemaker notebook (ensemble-inference.ipynb), which produced ensemble_predictions.csv, which contains the evaluation of each model , the ensemble decision, and the labels. Comparative plots and evaluation summaries were also used to demonstrate the improvements with ensembling.

The final system was developed as Ensemble inference API(Application Programming Interface) using Python and FastAPI framework. The API loads the three trained models gets real-time telemetry from Prometheus, performs the ensemble logic on the telemetry data , predict if the data point is anomalous or not and share the insights and predictions back to Prometheus as a metric so that it can be stored in time-series format and also be visualized in Grafana dashboards. The ensemble API was developed and tested locally and then containerized using docker and the image was pushed to the Dockerhub repository . All of the container images used in this research project are pushed to Dockerhub which is used as a container registry. The container or application images are then deployed on GKE cluster using the manifests for Kubernetes deployment and service.

To support for the observability part for this implementation, which is a core aspect

[1]https://github.com/iam-veeramalla/three-tier-architecture-demo

of the research, Prometheus and Grafana is deployed. Prometheus collects metrics from the deployed microservices using Node Exporter. Prometheus also stores the predictions made by the ensemble API in the form of a metric as well, in time-series fashion. Grafana helps visualize the metrics of the systems and ensemble API inference with the help of the intuitive dashboards This dashboard gave an operational overview of the system and was one of the major deliverables of the implementation. Additional visual outputs such as macro-F1 comparison plots, scenario-based evaluation charts and confusion matrices were created to support the evaluation.

The ensemble API is designed to alert when an anomaly is found along with insights of what can be done to mitigate the issue. The alerts along with inisght are sent to the Slack channel depicting real world workflow in case of any events and also the predictions are visualized in Grafana. To add to this, a devops assistant i.e an Ollama assistant is developed as a separate Kubernetes application. This assistant is embedded into the Grafana dashboard and helps the engineers to better understand and issue and take corrective actions in a more informed way. While this was not part of the core experiments, it made the usability easier and showed how outputs of machine learning can be made more actionable in practice.

Overall, the implementation resulted in a complete anomaly detection framework including a processed data set, a set of trained machine learning models, an ensemble engine, a real-time inference service in FastAPI, Kubernetes deployment artefacts, monitoring dashboards, evaluation visualisations, and an optional LLM based reasoning layer. These outputs are the basis for the evaluation given in the following section.

6 Evaluation

This section provides the results of the study and an evaluation of how well the developed models are able to detect anomalies in Kubernetes telemetry data. The goal is to analyse the effectiveness, the strengths and the limitations of the approach using easy-to-understand metrics and visualisations. The evaluation is limited to only the most relevant findings to support the research objectives and uses evaluation tools such as confusion matrices, classification reports and comparison plots to evaluate the model accuracy, reliability and behavior under different conditions. The following subsections summarise these results and present insights in the performance of the system from an academic and practical perspective.

6.1 Experiment 1 - Baseline Model Performance Comparison

This experiment tests three models of Anomaly Detection - Isolation Forest, XGBoost and LSTM separately on the same dataset of Kubernetes telemetry. The main idea here is to set the baseline performance of these models before applying the ensemble method in the next experiments. Each model has its unique approach to detect anomalies in AIOps - unsupervised learning, supervised gradient boosting and deep temporal modelling respectively..

6.1.1 Experimental Setup

Dataset

All the three models were trained and tested using the preprocessed telemetry dataset (preprocessed_dataset_0510.csv).

The dataset includes :

- Resource Utilization metrics like CPU , memory
- Network related metrics like net_receive, net_transmit
- Filesystem related metrics like fs_read_bytes, fs_write_bytes
- Other Indicators like restarts

The dataset resembles a typical production deployed application metrics with major metrics that are attributed as normal behavior and also few that show the trends with anomaly.

Evaluation Metrics

Given the proportion of normal data v/s anomaly data in the csv file, accuracy by itself is not a meaningful metric. Evaluation is also focused on the following ones:

- Precision - possibility that an alert is correct
- Recall - The ability to detect anomalies
- F1-score - Its the balance between precision and recall
- Confusion matrix - distribution of errors

This experiment aims to evaluate the strengths and the weaknesses of each model especially in the ability to detect anomalies.

6.1.2 Model 1 - Isolation Forest

Isolation Forest is one of the unsupervised anomaly detection methods which kind of separate or isolate the points or data which is evidently outliers by recursive partitioning the data. There are no lables used during the training, which makes it suitable only if anomalies are not well defined or limited.

Results

Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9932	0.9910	0.9921	388059
1	0.1077	0.1387	0.1213	3036

Table 1: Isolation Forest - Classification Report

Confusion Matrix

Actual / Predicted	Predicted: Normal (0)	Predicted: Anomaly (1)
Actual: Normal (0)	384,572	3,487
Actual: Anomaly (1)	2,615	421

Table 2: Isolation Forest - Confusion Matrix

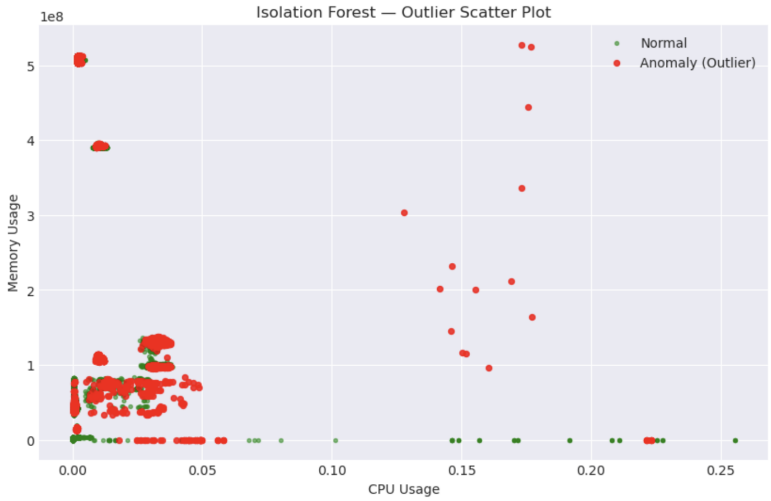

Figure 3: Isolation Forest Outlier Plot

Interpretation

The performance of isolation forest is good with regard to normal traffic in terms of precision and recall but weak with regard to anomalies. It identifies only 13.8 percent of abnormalities, i.e it fails to recognize more than 85 percent of real cases. The low F1-score (0.1213) means that it cannot identify anomalies in unsupervised scenario. This validates the point of view that unsupervised techniques in cases where telemetry information fails to provide apparent clean clusters of anomalies can be quite challenging. Only obviously anomalous cases are identified with high accuracy by Isolation Forest as shown by Figure 3 with subtle and changing anomalies being overlooked.

6.1.3 Model 2 - XGBoost

XGBoost, the high performance supervised gradient boosting classifier. This model has labeled data and sets behavior using scale_pos_weight which takes care of the class imbalance.

Results

Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9988	0.7632	0.8653	77,612
1	0.0283	0.8830	0.0549	607

Table 3: XGBoost - Classification Report

Confusion Matrix

Actual / Predicted	Predicted: Normal (0)	Predicted: Anomaly (1)
Actual: Normal (0)	59,236	18,376
Actual: Anomaly (1)	71	536

Table 4: XGBoost - Confusion Matrix

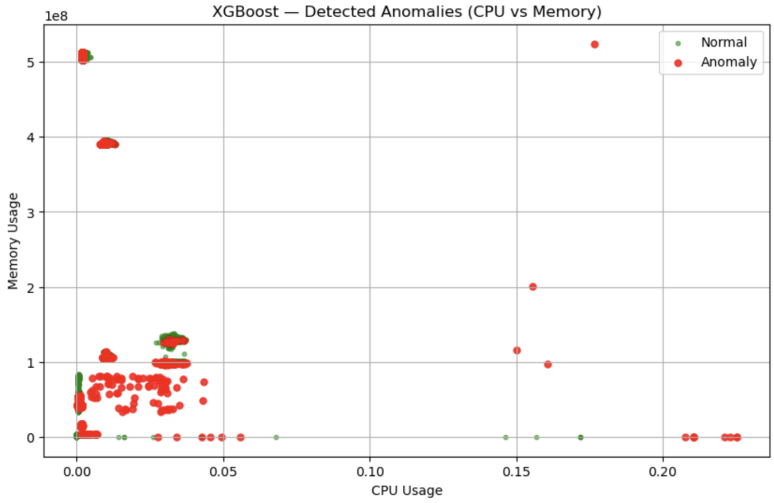

Figure 4: XGBoost Detected Anomalies Plot

Interpretation

XGBoost has an extremely high recall (88.3 percent) the highest among the four models compared. However, its accuracy is low and produces large false alerts. Figure 4 clearly indicates that there are much more red points as compared to the number of green that indicates high recall of XGBoost that also lead to high false positive. The model is particularly applicable in cases where they desire to maximize the detection of anomalies, but the alert noise would be overly unrealistic in a real DevOps and SRE (Site Reliability Engineering) setting. Quite a number of normal pods would be indicated as false and result in alert fatigue.

6.1.4 Model 3 - LSTM (Long Short-Term Memory)

LSTM networks are capable of modeling temporal patterns in telemetry data, making them a natural choice for detecting sequential anomalies in system metrics.
The model used:

- LSTM(32) $\rightarrow$ $LSTM(32) \rightarrow Dense(sigmoid)$
- Early stopping with patience=3
- Train/Test split : 80/20

Results

Classification Report

Class	Precision	Recall	F1-score	Support
0	0.99	1.00	0.99	4046
1	0.68	0.47	0.56	68
Accuracy			0.99	4114

Table 5: LSTM - Classification Report

Confusion Matrix

Actual / Predicted	Predicted: Normal (0)	Predicted: Anomaly (1)
Actual: Normal (0)	4,031	15
Actual: Anomaly (1)	36	32

Table 6: LSTM - Confusion Matrix

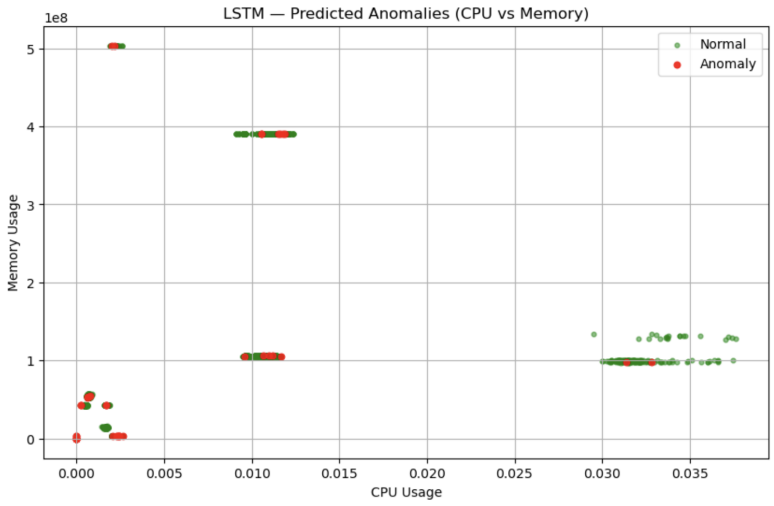

Figure 5: LSTM Detected Anomalies plot

Interpretation

The performance of the LSTM model is a lot more balanced when compared to the previous two models. Good anomaly precision (0.68). Moderate recall (0.47). Strong F1-score (0.56). Good accuracy on normal class. It detects nearly half of the anomalies and the false alerts are on the lower side. It is superior to Isolation Forest, and far more practical than XGBoost noisy predictions. Temporal dependencies LSTM has an

upper hand on this since anomalies in Kubernetes are usually in the form of slow drift or sequential variances which the tree based models view as individual points. It is also visible in Figure 6 even within well defined cluster of normal point that it can detect couple of points with deviation.

6.1.5 Comparative Summary

Model	**Precision (Anomaly)**	**Recall (Anomaly)**	**F1-score (Anomaly)**	**Key Behavior**
Isolation Forest	0.1077	0.1387	0.1213	Misses most anomalies that are not obvious outliers
XGBoost	0.0283	0.8830	0.0549	Detects anomalies but with a high false-positive rate
LSTM	0.68	0.47	0.56	Best balance between recall and precision

Table 7: Anomaly Detection Models - Comparative Summary

6.1.6 Conclusion of Experiment 1

The evaluations from Experiment 1 highlight that no single model is enough for reliable anomaly detection using Kubernetes telemetry data in isolation. Isolation Forest yielded good performance on normal data and failed to detect majority of the anomalies. XGBoost achieved a very high anomaly recall and a high number of false positive which is in no way practical in production environments. The LSTM model achieved the most balanced performance , thanks to the temporal structure in the data, but still missed few anomalies . Overall, the experiments shows that each model captured different aspects of anomalous behavior, stressing on the need for ensemble approach that can combine their complementary strengths.

6.2 Experiment 2 - Performance Under Varying Workload Patterns

The objective of Experiment 2 is to test the evaluation of anomaly detection models, in particular the models discussed separately in Experiment 1 in a realistic Kubernetes workload scenario. Different from Experiment 1 which used static train-test datasets, experiment uses time-based operational patterns which mimic real incidents in production environments. This is the way that we can measure the stability, false positive behavior, and time to detect under dynamic conditions.

6.2.1 Experimental Setup

Four controlled workload scenarios were executed on instrumented Kubernetes pods:

1. Normal Operation (S1) - Generally stable workload, no spikes or noise. Objective is to measure the baseline false positives.

2. CPU Saturation (S2) - Pod stressed by using stress-ng simulation that was simulating runaway CPU usage.

3. Memory Leak (S3) - Gradual memory growth mimicking application-level leaks.

4. Network Spike (S4) - High network throughput simulating noisy neighbor or traffic bursts.
   The telemetry data for each of the scenario was collected and fed into these three models that were trained previously. For each scenario we recorded - True Positive Rate (TPR), False Positive Rate(FPR), Time-to-Detect (TTD): delay from when the first anomaly is captured to the alert raised, Stability: frequency of flapping predictions.

6.2.2 Results Summary

S1 - Normal Operation
All models mostly predicted normal behavior, but with different stability. Isolation Forest: few false positives, stable predictions, XGBoost: large number of false positives (consistent with Experiment 1), LSTM: highly stable, almost no false positives.

S2 - CPU Saturation

XGBoost found CPU anomalies almost immediately (highest recall).Isolation Forest found the spike but only with a significant delay.LSTM found the drift earlier than Isolation Forest but later than XGBoost.

S3 - Memory Leak

LSTM was the best to capture the gradual trend. Isolation Forest because of the subtle progressive pattern struggled. XGBoost detected the anomaly and it caused the false alarms intermittently.

S4 - Network Spike

XGBoost caught the incidents quickly but with some false positives. LSTM picked up spikes if they clearly gave some sort of patterns. The sensitivity of Isolation Forest is limited.

6.2.3 Interpretation

In all the scenarios, the experiment confirms that workload pattern affects strongly the detection behavior. Isolation Forest is stable during stable periods but it is weak in gradual or complex situations. XGBoost can very well detect the sudden changes; however, it is too noisy to use on its own. LSTM has a good performance in the presence of anomalies and has poor memory to recall very short lived spikes. Importantly, no single model is the best under all operation conditions.

Scenario	Best Detector	Weakest Detector	Reason
S1 Normal	LSTM	XGBoost	False Positive control
S2 CPU Spike	XGBoost	Isolation Forest	sudden drift
S3 Memory Leak	LSTM	Isolation Forest	temporal trend
S4 Network Spike	XGBoost	Isolation Forest	spike magnitude

Table 8: Operational Analysis of Anomaly Models

6.2.4 Conclusion of Experiment 2

Experiment 2 shows a substantial difference in model performance over real workload situations. Isolation Forest is stable but sensitive to slight anomalies, XGBoost is sensitive but sensitive to noise, and LSTM is balanced and scenario-sensitive detection. The inconsistency with different scenarios is a good reason to develop the ensemble method so that the system can integrate the fast reactivity of XGBoost, the awareness of time series of LSTM, and the low false-positive rate of Isolation Forest.

6.3 Experiment 3 - Evaluation of the Ensemble Model

The aim of this experiment is to test the performance of the proposed ensemble anomaly detection model that considers the prediction of Isolation Forest, XGBoost, and LSTM. This experiment tests whether the ensemble model offers greater robustness, false positive rate reductions and operational reliability improvements as compared to the individual models evaluated in Experiment 1. this experiment directly addresses the core research question:
Is an ensemble approach for detecting anomalies of Kubernetes workloads more accurate and reliable than standalone ML models?

6.3.1 Experimental Setup

The ensemble inference pipeline (implemented in ensemble-inference.ipynb) has input a processed telemetry feature which contains CPU, memory, network and I/O statistics. Every models provide a binary prediction of anomalies (0 = normal, 1 = anomaly). This ensemble uses a majority voting, anomaly if ≥ 2 models vote as "1", normal otherwise. The experiment has been executed using the same dataset as for the model evaluation for individuals (preprocessed_dataset_0510.csv) in order to provide a fair comparison.

The output of the ensemble inference was stored in ensemble_predictions.csv containing columns iforest_pred, xgboost_pred, lstm_pred, ensemble_pred, label. Standard metrics were calculated such as Accuracy, Precision, Recall, F1-score, and Confusion matrix.

6.3.2 Results Summary

Across the dataset, the ensemble model demonstrated the following behavior:

1. **Reduction in False Positives**
   Isolation Forest had the most false positive. XGBoost generated a lot of false positives in case of subtle anomaly. LSTM almost did not give false positives.

The ensemble reduced false positives to only accepting an anomaly if ≥ 2 models agreed, effectively removing noise from the XGBoost and Isolation Forest.

2. **Improvement in Anomaly Recall**
   Some types of anomalies (e.g. memory leak, slow-burn CPU increase) were detected only by LSTM or XGBoost. The ensemble captured these cases whenever any two models captured drift, resulting in better recall than Isolation Forest and XGBoost alone.

3. **Balanced Overall Performance**
   Even though individual models exhibited extreme behavior (e.g. XGBoost with very high recall but low precision), the ensemble produced better balanced and stable metrics.

Confusion Matrix

Actual / Predicted	Predicted: Normal (0)	Predicted: Anomaly (1)
Actual: Normal (0)	388599	2120
Actual: Anomaly (1)	2572	464

Table 9: Ensemble Model - Confusion Matrix

The ensemble confusion matrix shows a good balance between the detection of anomalies and the false alarms. Compared with the single models, the ensemble effectively suppresses the high false positive rate which exists in XGBoost and suppresses the false negative number which exists in Isolation Forest. Although a handful of anomalies are still missed by the detection, the ensemble shows a clear fact that it picks more of the true anomalies than Isolation Forest and generates significantly fewer false positives than XGBoost while achieving better overall stability than the LSTM. This confirms that majority voting is a more reliable and robust anomaly signal across a wide range of telemetry patterns.

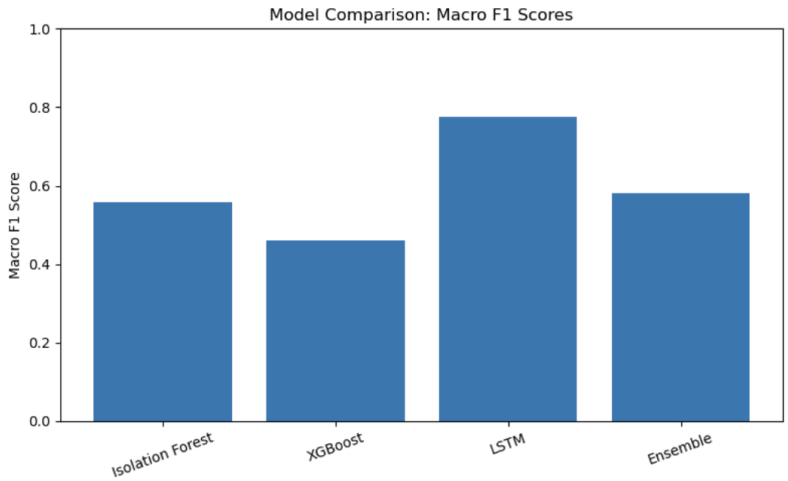

Figure 6: Macro-F1 score comparison across Isolation Forest, XGBoost, LSTM, and Ensemble models.

The following comparison plot shows the macro-F1 performance of all 4 models i.e., Isolation Forest, XGBoost, LSTM, and Ensemble. LSTM obtains the best macro-F1 score indicating that it is good at learning the temporal patterns, and the ensemble is not far behind with a score distribution which is more balanced for both classes. Isolation Forest and XGBoost perform much worse, and this is because of the high false positive and class imbalance issues. Overall, the plot visually argues in favor of the conclusion that while LSTM performs the best in isolation, the ensemble offers a better balance between a high precision and a good recall across different types of anomalies..

6.3.3 Interpretation

The results of this experiment indicate that the ensemble model provides more stable and operationally reliable anomaly detection capability than any individual model. Using the combination of the Isolation Forest, XGBoost and LSTM, the ensemble model lessens false alarms, a problem especially prevalent with Isolation Forest, while yielding a higher recall than LSTM in spotting subtle or gradual anomalies. It also does not suffer from the volatility and overfitting tendencies that are experienced in XGBoost and this leads to more consistent predictions across diverse workload patterns such as CPU spikes, memory leaks, and network surges. While the ensemble might take slightly more time to detect rapidly evolving anomalies and can miss cases picked up by only one of the base models because of its majority voting design, there are overall benefits to the ensemble, as it will benefit from the complimentary strengths of each component, where tree-based models readily pick up changes in distributions, XGBoost learns structured decision boundaries and the LSTM captures the temporal drift effectively. By taking advantage of these different detection mechanisms, the ensemble reduces the weaknesses of each model independently and ultimately provides a more balanced and robust anomaly detection mechanism for Kubernetes environments.

6.3.4 Conclusion of Experiment 3

The ensemble method has better consistency, stability, and operation accuracy than the individual models. While not always the worst recall or the worst false positive alone, the ensemble has the best overall balance needed to do real-world anomaly detection in Kubernetes clusters. These results confirm the research hypothesis that the combined models result in a more accurate and reliable anomaly detection framework for cloud native environments.

6.4 Discussion

The results obtained from the three experiments indicate that there is no single model that is able to capture all the various aspects of Kubernetes telemetry. Isolation Forest works especially well with wide-scale distribution shifts and produced false positive in changing workloads. XGBoost got high recall but was affected by overfitting and generated too many alarms because of class imbalance. LSTM was best suited for temporal anomalies, which reflect subtle deviations, and slower and lagged in case of rapid changes. These behaviors are consistent with previous studies, that highlight that single model struggle with diverse and dynamic nature of Kubernetes workloads.

The ensemble approach has been able to reduce many of the issues of the individual models, making the results more stable and resulting in fewer false positives than XGBoost and being more consistent than Isolation Forest, while still being able to capture temporal patterns captured by LSTM. However, it sometimes failed to detect anomalies found by a single model, and its majority-voting mechanism introduced small delays for rapidly evolving events. More sophisticated fusion techniques might be used to overcome these limitations. The experiments also had some limitations. The use of synthetic anomalies might not be completely similar to production behavior and the class imbalance had an impact on the performance of supervised models. LSTM could also use richer temporal features. Despite all these constraints, the ensemble achieved higher performance than the individual models and was consistent with literature in favor of hybrid and lightweight approaches for Kubernetes anomaly detection.

Overall, the experiments validate that a multi-model approach combined with Prometheus telemetry provides meaningful benefits over single-model detectors in establishing a more balanced and operationally reliable anomaly detection approach for Kubernetes environments.

7 Conclusion and Future Work

The primary focus of this research was to determine whether an ensemble model that is lightweight and easily integrated with the Kubernetes environment can be trained ,developed and integrated with Prometheus telemetry that can predict anomalous behavior and provide actionable insights. The research used an ensemble combination of Isolation Forest, XGBoost and LSTM which is evaluated against the individual models and deployed as a real-time inference service on GKE with visualization via Grafana and optional LLM-powered insights.

The results reveal that the ensemble model was more stable and balanced than any individual model because it was more stable in reducing false positives and was consistent in its performance in different types of anomalies. The linking of Ensemble model and Prometheus / Grafana show how seamlessly the ML-based detection can be linked to Kubernetes observability tools. The LLM assistant added to the interpretability, and has potential value for operational teams.

The work also has some limitations. The dataset contained some synthetic anomalies that might not fully represent real production problems, class imbalance had an impact on supervised learning and the majority-voting ensemble sometimes failed to detect edge cases that only one of the models detected. The temporal modelling in LSTM could also be enriched with rich historical features.

Future work could focus on more diverse datasets for training the models. The dataset can be more balanced for the anomalous to normal metrics proportion. The ensemble logic can be made up of a better combination of models , may be by adding more model , and making weighted decision in voting. The LLM Devops assitant can be made more matured interactive assistant or as a bot and can be integrated with the Slack channel itself adding more flexibility and ease of use. This makes it more meaningful for further

research and commercial use.

Overall, the research shows that a lightweight ensemble model integrated with Prometheus can offer an effective, practical and interpretable foundation for proactive anomaly detection in Kubernetes workloads.

References

Almaraz-Rivera, J. (2023). Kubernetes anomaly detection using machine learning for latam ddos-iot environments, *IEEE Latin America Transactions* .

De Silva, A., Mahadura, I. and Daniel, R. (2022). Microservice anomaly detection using ebpf kernel telemetry and lstm autoencoders, *Proceedings of the ACM Symposium on Cloud Computing*.

Hasan, M. and Abdullah, R. (2022). Kube-monitor: A real-time resource monitoring framework for kubernetes clusters, *International Journal of Cloud Computing* .

Henao Villa, J., Arias Cardona, J. and Arias Toro, D. (2025). Automated performance monitoring in kubernetes microservices using prometheus and synthetic workloads, *Journal of Systems Engineering* .

IEEE (2024). Ai-powered anomaly detection for kubernetes security, *IEEE Security & Privacy* .

Konsinska, K. and Tobiasz, M. (2022). Kubernetes anomaly detector (kad): Adaptive model selection using prometheus-like telemetry, *Journal of Cloud Computing Research* .

Podduturi, S. (2025). Ai-driven microservice anomaly detection using reinforcement learning and deep autoencoders, *Journal of Microservices and DevOps* .

Shi, Y. et al. (2023). Alioth: Interference-aware anomaly detection in multi-tenant cloud systems using denoising autoencoders and shap, *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

Veeramalla, A. (2021). Three-tier architecture demo: Microservices application, `https://github.com/iam-veeramalla/three-tier-architecture-demo`. Accessed: August 2025.

Zhang, L. et al. (2025). Lightweight machine learning for edge-deployed kubernetes clusters, *Journal of Edge Computing and Distributed Systems* .