



Original Article

A Machine Learning-Based Framework for Predictive Performance Optimization and Fault Detection in Cloud-Native Distributed Computing Infrastructure

DR. A. Basheer Ahamed

Assistant Professor, Department of Computer Science, Jamal Mohamed College (Autonomous) Tiruchirappalli.

Received On: 01/06/2026

Revised On: 28/06/2026

Accepted On: 08/07/2026

Published On: 19/07/2026

Abstract - Scales, resilient and highly available applications depend on cloud-native distributed computing infrastructure. The complexity of interactions between microservices, containers, orchestration platforms, workloads and shared resources, however, poses a significant challenge in the areas of optimal performance and early fault detection. In this paper, a novel machine learning-based framework that focuses on predictive performance optimization and fault detection in cloud-native distributed computing infrastructure is presented. The proposed framework is based on continuous infrastructure monitoring, telemetry data collection, data preprocessing, feature engineering, machine learning-based performance prediction, resource optimization, anomaly detection, fault classification, and intelligent decision-making, all of which are part of a closed-loop system. The heterogeneous telemetry data (CPU and Memory utilization, network activity, input/output operations, response latency, throughput, error rate, logs, distributed trace, orchestration event) is analyzed to understand the patterns of systems behavior and forecast the future state of the systems. The prediction layer forecasts the behaviour of workloads, resource requirements and potential performance degradation which can then lead to proactive decisions on how to handle workloads, including the possibility of autoscaling, reallocating resources or redistributing workloads. Concurrently, the fault and anomaly detection engine detects anomalies, categorizes failures and helps to conduct early failure mitigation and root cause analysis. The experimental evaluation shows the potential of the proposed approach to improve the accuracy of the predictions, to reduce the response latency, to minimize the breach of the service level agreements, to optimize the utilization of the resources and to improve the fault-detection performance, over the conventional reactive approaches. The framework is designed to be scalable and adaptable to create intelligent, self-adaptive and resilient cloud-native distributed computing environments.

Keywords - Distributed computing infrastructure, Machine learning, Predictive performance, Fault detection, Anomaly detection, Resource optimization, Kubernetes.

1. Introduction

Cloud-native distributed computing infrastructure has emerged as a key cornerstone of today's digital services, helping to power applications that demand scalability, flexibility, high availability, and speed to market. Container, microservices, Kubernetes, service meshes, and distributed storage platforms are some of the technologies that allow organizations to deploy and manage complex applications across a range of geographies in a distributed computing environment. [1] But these infrastructures have dynamic properties which pose huge challenges for operation as well. App performance and service reliability can suffer if workloads increase or decrease quickly, resources are competing, network delays occur, configuration changes are made, or software or hardware failures occur. The traditional infrastructure management methods can be reactive, discovering issues after they become a performance problem or system failure.

The machine learning approach is one that is promising in shifting the cloud infrastructure from a reactive monitoring to a proactive and predictive decision-making process. Machine learning models can monitor several sources of telemetry, such as CPU usage, memory consumption, network traffic, latency, throughput, error rates, and application logs, and uncover hidden patterns and anomalies from the vast amount of data. Predictive models can also predict future requirements of the resources and possible performance constraints, allowing an infrastructure manager to implement preventive measures before a service level goal is breached.

The proposed methodology using machine learning in this research is designed for predictive performance optimization and fault detection in the cloud native distributed computing system. It brings together real-time monitoring, intelligent data analysis, anomaly detection, fault prediction, and adaptive optimization mechanisms in a single-architecture framework. The system can take advantage of predictive knowledge to enable features like dynamic resource allocation, workload balancing, container scaling and early fault mitigation. The idea behind the proposed approach is to increase the reliability of infrastructure, optimizing the use of resources in order to

minimize the downtime of operations and help in the creation of intelligent, self-adaptive and resilient cloud-native computing environments.

2. Related Work and Literature Review

2.1. Cloud-Native Distributed Computing Infrastructure

Cloud-native distributed computing infrastructure has become an architectural trend for modern enterprise applications, and it has become a dominant technique for creating and running applications in the cloud. It's defined by the use of containers, microservices, orchestration platforms, automated deployment pipelines, elastic resource provisioning, and DevOps practices. [2] Cloud-native systems are different from the monolithic applications or relatively static virtual-machine-based systems. These services interact across clusters of physical or virtual computing resources, via APIs, service meshes, messaging systems, and distributed networking mechanisms. This architectural style offers numerous benefits with regards to scalability, portability, fault isolation and deployment flexibility.

On the other hand, cloud-native applications come with significant challenges in terms of complexity in operations due to their distributed and dynamic nature. There are many factors that affect the performance of an application, such as peak and off-peak workloads, shared resource utilization, placement of containers, network latency, service dependencies, change in service configuration, and competition for shared infrastructure resources. If one microservice has a performance issue, it can spread to other services that are dependent on it, leading to an impact on the application as a whole. Traditional monitoring methods using mostly thresholds then may not be enough to detect more complicated or slowly developing performance issues.

To overcome this limitation, modern cloud-native observability combines several aspects of an operation to provide insights. Metrics give quantitative data on infrastructure and application behavior, logs give detailed data on system and application activities, and distributed traces give system interactions between services during individual transactions. These data sources complement each other to yield a more complete picture of the system's behavior. Intelligent anomaly detection and reliability analysis are increasingly employed in various contexts such as Kubernetes clusters, microservice architectures, edge environments and service meshes. These methods can help detect unusual activities that could not be detected using individual monitoring signals.

Microservice based applications create fault diagnosis, especially hard due to their distributed nature. User-facing service failure can happen due to CPU saturation, memory pressure, network congestion, failures of other services, containers crashing, misconfigurations or cascading failures among multiple services. Thus, the management of abnormal conditions is not enough for effective infrastructure management, it should also learn the relations between the infrastructure components and application services. The

proposed framework is based on this view, which takes the cloud-native infrastructure as a connected and continuously evolving system. Combines infrastructure metrics, application telemetry, logs, traces, deployment data and orchestration events to facilitate predictive performance analytics and intelligent fault detection.

2.2. Machine Learning for Performance Prediction

In the context of cloud computing, machine learning technologies have become a hot topic for predicting workload behavior, resource usage, and application performance. [3] Traditional infrastructure management methods may be based on historical averages or preset thresholds for when more resources need to be expended. These methods can fail if tasks are changing frequently or if they have irregular task patterns. Machine learning models can be trained with the historical and real-time telemetry to discover the relationships between workload characteristics, infrastructure usage, and application performance helping to predict future infrastructure states before resources are saturated or performance drops.

Numerous techniques are investigated to predict cloud workloads. If the workload has relatively stable temporal behavior, the following statistical forecasting methods are often used. Regression models can be used to estimate relations between the workload demand and the consumption of resources, and support vector machines, decision trees, random forests and ensemble learning methods can be used to model more complex nonlinear relations. Recently, deep learning approaches have been considered to capture long-term temporal dependencies and multi-dimensional interactions between the CPU utilization, memory usage, network traffic, request rate, storage activity and application latency.

There are several important activities in cloud management which are supported by predictive models. [4] Workload demand can be forecasted and proactive autoscaling can be used to allocate more computing resources, redistribute workloads, balance traffic, migrate workloads, or begin before performance goals are being breached. Likewise, the predictions of latency and throughput can help identify potential bottlenecks in advance. Specifically in a cloud-native environment, automatic scaling actions could potentially take time to start containers, allocate resources, or re-distribute traffic for the application.

Despite these benefits, the challenge of performance prediction remains as the cloud workloads are heterogeneous, multidimensional and constantly evolving. If workload patterns change from those used for training, or if the infrastructure configuration is changed, then the accuracy of the model may drop. Besides the classical prediction metrics (Mean Absolute Error, Root Mean Square Error, coefficient of determination), operational aspects like prediction latency, model drift, generalization across services and the impact of incorrect predictions play a part. Thus, the presented framework focuses on learning models which are

continually updated, combining historical data and real-time observations to make predictions applicable for actual optimization decisions.

2.3. ML-Based Resource Optimization

Resource optimization is one of the key aspects of machine learning, aimed at optimizing computing resources and ensuring that the allocation and utilization of these resources can be optimized while maintaining the service-level objectives, reliability, and performance of the applications. Cloud-native architectures require the provisioning of resources like CPU, memory, storage, network bandwidth and computing nodes, to be dynamically provisioned to support applications that have variable workloads. [5] Traditional methods often rely on rules that are manually programmed or a fixed utilization level. These mechanisms are easy to set up, but can cause poor scaling response times, underuse of resources, overuse of resources when traffic drops, or lack of resources when traffic spikes.

Machine learning offers a more adaptive solution, with learning patterns from operational data. Supervised learning models are able to forecast the future resource requirements when considering the use of resources in the past. Such forecasts can help to make proactive capacity allocation before reaching the critical levels. Dynamic scaling and scheduling has also been explored by reinforcement learning: an optimization agent is able to learn resource-management policies by interacting with the computing environment. Alternate methods use ensemble learning and deep neural networks to produce models of intricate relationships between workload intensity, infrastructure state, resource usage and application performance.

In cloud-native infrastructure, intelligent resource optimization can help inform decisions concerning where to place pods, how to scale containers, which nodes to use, how to consolidate workloads, where to move services, and how to route traffic. But, optimizing cannot be just based on the expected demand of the resources. In real-world cloud scenarios there are issues of scaling delays, cold-start time, resource quotas, affinity/anti-affinity rules, service dependencies, infrastructure capacity, operational cost, and service-level agreements. An optimization strategy has to therefore consider the possible performance gain of an action and its resource and operational cost.

The proposed framework is a closed loop approach towards optimizing the resources. Machine learning models take information from the telemetry and can foresee workloads or performance issues. An optimization layer, based on these predictions, decides what actions should be taken, including scaling the resources, distributing workloads, or changing the placement of services. The impact of these changes on the performance of the system is then observed and fed back into the learning process. This feedback mechanism enables the framework to continually adjust its predictive and optimization functions over time, helping to create proactive and self-adaptive infrastructure management, instead of only reactive resource control.

2.4. Fault and Anomaly Detection

Reliability and availability of cloud-native distributed computing systems are fundamental requirements and thus fault and anomaly detection is of critical importance. IT failures can occur in these types of environments, such as failures in infrastructure components, software services, network connections, configuration changes, resource exhaustion, or dependencies between distributed applications. [6] Anomalous behaviour can manifest itself in many different ways, such as CPU usage or memory consumption that is not what you expect, latency spikes, error rates, container restarts, failed service requests, network outages, or oddities in application logs. Cloud native systems produce lots of operational data, which can be very heterogeneous and is difficult to analyze manually and using only simple thresholds.

One of the ways to detect abnormal behavior is through a technique known as machine learning, which maps out the characteristics of a normal system operation and then learns how to recognize the characteristics of a faulty system operation. With an existing set of labeled training examples, supervised learning methods can be used to categorize known fault types. A complete description of the fault, however, is often hard to come by as incidents in real production environments can be rare and new. Cloud infrastructure monitoring is thus a particularly suitable application for unsupervised and semi-supervised methods. Methods like clustering, isolation-based anomaly detection, principal component analysis, and one-class classification may be able to detect observations that are far from typical operational behavior patterns when there is not a large data set of labeled failures.

A further advantage of the deep-learning approaches is the ability to capture complex temporal and multivariate relationships, which further enhances the capabilities of anomaly detection. Autoencoders can be trained to reconstruct normal system behavior, and deviations from it can be detected by the reconstruction errors. Sequences of telemetry observations can be used to recognize changing performance issues using recurrent neural networks and related temporal models. When it comes to modeling relationships between microservices and infrastructure components, graph-based models are well suited and relevant to detecting anomalies and fault propagation in service dependency networks. Nevertheless, these methods may require substantial training data and computational resources, while their internal decision processes can be difficult to interpret.

3. System Model and Problem Formulation

3.1. Cloud-Native Infrastructure Model

The proposed system is designed to support a distributed computing system with multiple components distributed in the cloud, interconnected with each other. Infrastructure consists of computing nodes (physical or virtual), containerized services, microservices, orchestration platforms, networking infrastructure, storage infrastructure, and monitoring infrastructure. [7] Containers are deployed

and managed dynamically over the available nodes as per the workload and resource requirement. Services rely on distributed APIs, messaging and network to communicate, which creates dependencies that impact the overall performance and reliability of the system.

Infrastructure state is a moving set of observations of infrastructure resources, infrastructure workloads and infrastructure service-levels. Several components could be impacted by changes in workload intensity, container placement, node availability, network state or application configuration. Hence, the system model considers the cloud-native environment as a time-varying, interconnected system and continuously monitors data to help with performance prediction, resource optimization and fault detection.

3.2. Infrastructure Performance Metrics

A number of metrics at the system and application level are used to assess the performance of the cloud-native infrastructure. Some of the key metrics in the infrastructure are CPU utilization, memory usage, storage usage, network bandwidth usage, packet delay, and the availability of nodes. [8] Application level metrics are measures such as response latency, throughput, request rate, error rate, container restart frequency and service availability. These measures are used to describe the infrastructure at a particular moment in a multi-dimensional manner.

The state of the performance at time (t) can be represented by a feature vector (X_t) that includes the infrastructure and application metrics that are being monitored. By continuously observing (X_t), the system can learn about the temporal patterns, detect abnormal conditions and predict future condition of the system. These metrics also are the key features used by the machine learning models in prediction and anomaly detection.

3.3. Workload Model

The workload model is the fluctuating demand placed on the cloud-native infrastructure as it changes with time. The characteristic of the workload can be the number of incoming requests, the number of transactions processed, the number of concurrent users, the amount of data processed, and the frequency with which the services are called. These variables may vary greatly as a result of user activity, periodic processing jobs, application events, or sudden surges in traffic. Workload demand is thus modeled as a time-dependent process (W_t).

Recurring patterns, trends, and short-term fluctuations can be determined from historical workload observations. The machine learning model takes historical workload values and infrastructure telemetry as inputs and outputs the predicted workload value for the future $\hat{W}_{t+h}$ for prediction horizon (h). Workload prediction helps in making proactive decisions on scaling and allocation of resources, before a significant impact on their performance is observed as a result of their increased demand.

3.4. Performance Prediction Problem

The performance prediction problem is to predict the future behaviour of one or more infrastructure performance metrics based on historical and real-time observations. [9] The predictive model learns a function based on the past observed system states, workloads and telemetry measurements to predict a future performance value. This can be expressed as:

$$P^{t+h} = f(X_{t-k:t}, W_{t-k:t})$$

- P^{t+h} = predicted performance metric at future time t+h
- $X_{t-k:t}$ = sequence of infrastructure states from time t-k to t
- $W_{t-k:t}$ = sequence of workload observations from time t-k to t
- k = historical observation-window length
- h = prediction horizon
- $f(\cdot)$ = learned machine-learning prediction function

Prediction Error

$$et + h = Pt + h - P^{t+h}$$

- $et + h$ = prediction error at future time t+h
- $Pt + h$ = actual performance at time t+h
- P^{t+h} = predicted performance at time t+h

Mean Squared Error (MSE) Objective

$$LMSE = N1i = 1\sum N(P_i - P^{i})^2$$

- LMSE = mean squared error loss
- N = number of prediction samples
- P_i = actual performance value for sample i
- P^{i} = predicted performance value for sample i

The goal is to have the difference between the predicted performance and the actual performance be as small as possible without the prediction lag being too high for use in the operational system. The model can be used to forecast CPU usage, memory usage, response time, throughput, error rate or any other service-level indicator of performance depending on the chosen target variable. Such forecasts are used to predict potential constraints and to inform optimization decisions ahead of time.

3.5. Resource Optimization Problem

Resource optimization problem is one of determining an optimal allocation of computing resources to achieve application performance without over-provisioning. Define the resource allocation state (R_t) as the state of the CPU, memory, storage, network capacity, and available computing nodes. The optimization mechanism determines an action (A_t) based on the predicted workload and performance conditions; actions include scaling containers, reallocating

resources, redistributing workloads and adjusting service placement.

Resource Allocation State

$$R_t = [R_t^{(CPU)}, R_t^{(MEM)}, R_t^{(STO)}, R_t^{(NET)}]$$

Optimization Action

$$A_t$$

$$= [A_t^{(scale)}, A_t^{(alloc)}, A_t^{(place)}, A_t^{(route)}]$$

4. Proposed Machine Learning-Based Framework

4.1. Overall Framework Architecture

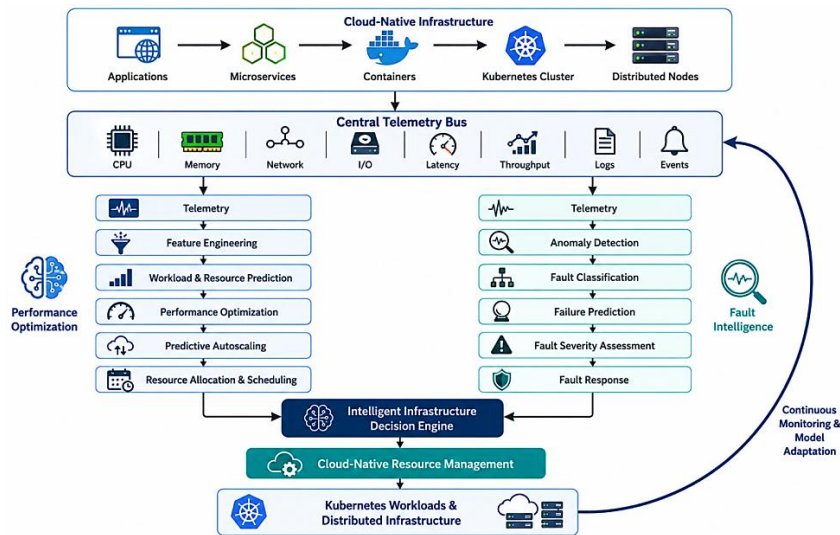


Figure 1. Proposed Machine Learning-Based Framework for Predictive Performance Optimization and Fault Detection in Cloud-Native Distributed Computing Infrastructure

The gathered telemetry data are run on two complementary intelligence pipelines. The performance optimization pipeline includes feature engineering, workload and resource prediction, performance optimization, prediction-based auto-scaling, and resource allocation and scheduling. [10] At the same time the fault intelligence pipeline performs anomaly detection, fault classification, failure prediction, fault severity assessment and response analysis. The results of both pipelines are combined in an

Architecture for a proposed intelligent control framework for cloud-native distributed computing infrastructure based on machine learning. The cloud-native layers are applications, microservices, containers, Kubernetes clusters, and distributed computing nodes. They are all interwoven components that constantly provide operational data, which are gathered via a central telemetry bus. The telemetry layer combines a number of infrastructure and application metrics such as CPU usage, memory usage, network activity, input/output operations, latency, throughput, logs, and system events. This is a centralized monitoring mechanism that will give a complete and up-to-date picture of the status of the distributed infrastructure.

intelligent infrastructure decision engine that decides on adequate management actions as a function of predicted performance conditions and faults detected. Such decisions are then applied to Kubernetes workloads and the underlying distributed infrastructure via the cloud-native resource management layer. This is a continuous feedback loop which allows ongoing monitoring and model adaptation, which helps to ensure the framework's predictive accuracy and efficient, reliable and self-adaptive cloud-native operations.

4.2. Infrastructure Monitoring and Telemetry Layer

The proposed Infrastructure monitoring and telemetry layer for continuously monitoring and observing the operational behavior of cloud-native workloads. The monitored environment consists of containers, application pods, microservices and other distributed application components that produce runtime data, such as logs, metrics and distributed traces. [11] The telemetry data is sent to specific log, metric and trace collectors that gather data from applications and infrastructure elements. The data collected is then pushed to a centralized monitoring repository where data is stored and indexed, time-series metrics are maintained for trend and performance analysis and distributed traces are preserved for request correlation and service dependency analysis.

The proposed machine learning framework will have its major data base in the centralized monitoring repository. The integrated monitoring data is sent to the monitoring and analysis layer which provides performance dashboards that show infrastructure health and important key performance indicators, and alerts management, which detects important events and creates operational notifications. The feedback loop in the architecture facilitates on-going observation and adaptive infrastructure management, with alerts and analytics provided in turn. The proposed framework integrates logs, metrics, and distributed traces to enable a single telemetry pipeline to deliver holistic, real-time operations data for future performance forecasting, anomaly detection, fault classification, and intelligent resource optimization.

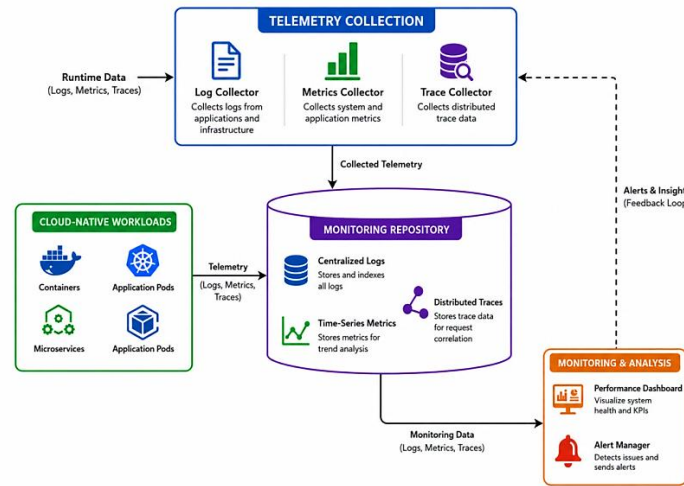


Figure 2. Infrastructure Monitoring and Telemetry Collection Architecture for Cloud-Native Workloads

4.3. Data Collection and Feature Engineering

The data preparation pipeline that was employed to process cloud-native telemetry data from multiple sources and render it into machine-learning-ready datasets. Raw telemetry data from various operational sources, such as logs, metrics, distributed traces and infrastructure events. [12] These data can be noisy, incomplete, inconsistent in format, and have unusual observations, so the data preprocessing step cleans up the data, removes noise, fills in

missing values, normalizes the data, and identifies the outliers. Data is then cleaned and then transformed via feature engineering which includes aggregation, temporal encoding, categorical encoding, dimensionality reduction, feature creation, and interaction analysis of relevant operational characteristics of the data. These processes map the substantial raw monitoring data into comprehensible models of the infrastructure and application activities.

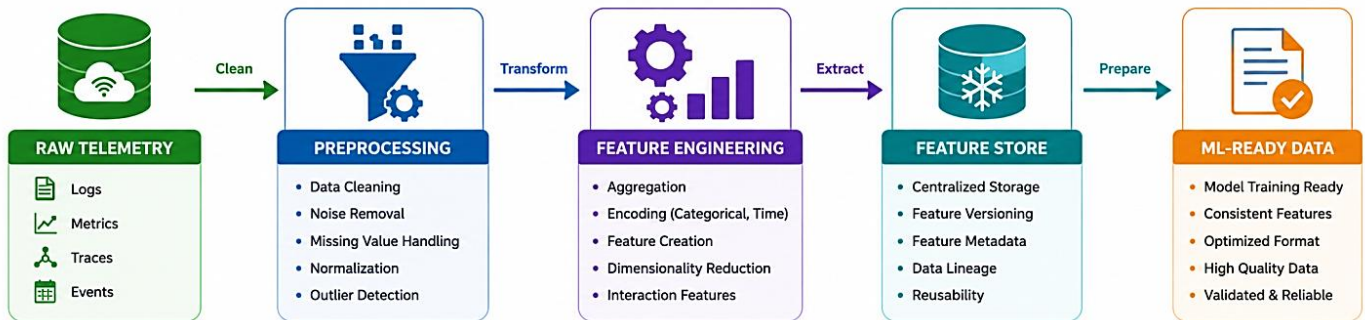


Figure 3. Data Collection, Preprocessing, and Feature Engineering Pipeline for Machine Learning-Based Cloud Infrastructure Analysis

The engineered features are stored in a centralized feature store with the following capabilities: feature versioning, feature metadata, data lineage, and feature reusability across machine learning tasks. This architecture allows for prediction of workloads, forecasting of workload performance, detection of workload anomalies, classification of workload faults, and optimization of workload resources throughout. Optimized, consistent and high-quality data layer that is ready for use in model training and real-time inference. The proposed framework strengthens the data quality, lessens complexity of training models, and ensures a solid basis for intelligent decision making modules of cloud-native infrastructure with the structured pipeline from raw telemetry to reusable features.

4.4. Machine Learning Prediction Layer

The machine learning prediction layer of the proposed framework. It starts with a structured feature vector that

holds information such as system metrics, history, context, and workload characteristics that come from the data collection and feature engineering layer. [13] These input features are given to machine learning models to learn the relationships between the conditions of the infrastructure, the behavior of the workload, and the performance of the system. The prediction layer contains a performance model to estimate future system level metrics, and a workload model to identify future workload characteristics. These models can be regression, ensemble learning, deep learning or other appropriate machine learning methods, depending on available data and deployment needs.

The outputs from the prediction models are further processed by a prediction engine to produce estimates of future performance and resource use of the system such as CPU, memory, input/output and network usage. The predictions are then translated into workload and resource

forecasts to give early warning of changing conditions of infrastructure. The forecasting engine allows the proposed framework to shift the management from being reactive toward proactive decision-making, where it can foresee the occurrence of resource shortages and performance

degradation before they impact services' quality. The expected outcomes can then be passed to the resource optimization and intelligent decision layers, which can then decide on actions like autoscaling, workload redistribution, resource allocation, and more.

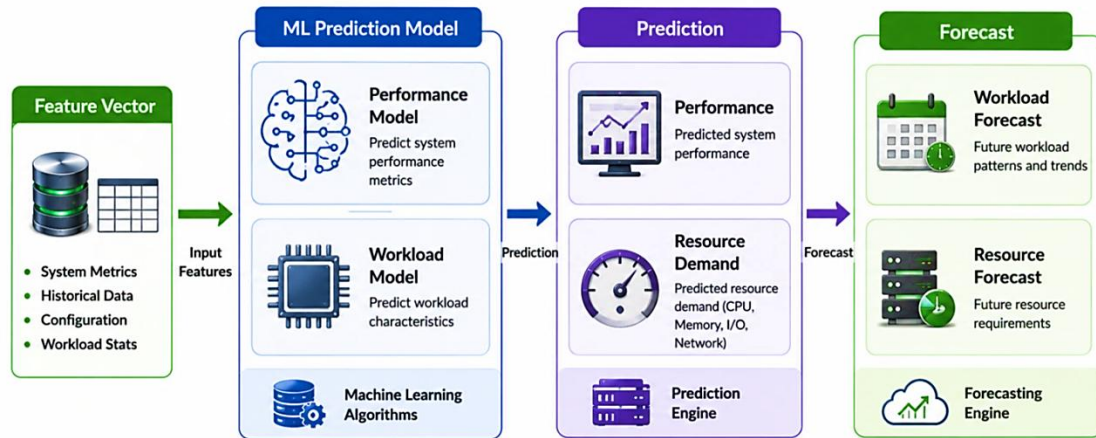


Figure 4. Machine Learning-Based Performance, Workload, and Resource Demand Prediction Layer

4.5. Performance Optimization Engine

The figure depicts an operational workflow of the proposed performance optimization engine. The process starts with predicted performance information generated by the machine learning prediction layer such as workload forecasting, performance estimation, demand forecasting and key performance indicator forecasting. These predictions are then provided to an optimization decision component that uses an analysis based on machine learning or artificial intelligence to consider potential actions. [14] The decision process can incorporate what-if simulations, optimization logic, operational policies, and infrastructure constraints to determine the most appropriate response to anticipated workload changes or performance degradation.

The resource allocation component then takes the optimization decision and implements the action, such as changing computing or memory resources, changing network or storage resources, or changing workloads scheduling and placement. A performance monitoring component is used to continuously monitor the resulting infrastructure state which tracks real-time metrics, KPIs, anomalies and overall system behavior. The feedback loop also provides updated monitoring data back to the predictive stage, allowing the framework to assess the outcomes of past actions and continually adapt and improve future optimization decisions. This closed-loop mechanism helps in the proactive, adaptive, and self-improving performance management of cloud-based distributed computing infrastructure.

4.6. Fault and Anomaly Detection Engine

Fault and anomaly detection engine constantly scans infrastructure metrics, application log, distributed traces, and service events to detect any anomalies in the behavior of the cloud-native system. Machine learning algorithms analyze the patterns behind abnormal resource use, latency spikes, error-rate rises, container failures, network issues, and

service dependency issues. Anomalies that are detected can then be sorted by fault type and severity, allowing the framework to attempt to distinguish between minor performance irregularities and fault situations that are critical. This layer is able to detect anomalies, classify faults, predict failures and rate the severity, which enables early warning in order to proactively mitigate faults and avoid cascading failures and service interruptions.

4.7. Decision and Resource Management Layer

The decision and resource management layer combines predictive performance analysis results, optimization recommendations, and fault intelligence for deciding on the right actions to take in the cloud-native infrastructure. The intelligent decision mechanism can trigger actions like container autoscaling, resource reallocation, workload redistribution, pod rescheduling, service recovery or fault-response procedures, depending on the current state and future behavior of the system. All of these are performed within the cloud-native orchestration and resource management environment, and continuous monitoring measures the effectiveness of these actions. The feedback is then fed back into the system to manage the state of the system and to aid adaptive decision-making, thus closing the loop and turning the proposed framework into a self-adaptive and resilient system for infrastructure management.

5. Machine Learning Methodology

5.1. Dataset Construction

The proposed framework uses data from a telemetry continuously collected from the cloud-native distributed computing environment. Multiple heterogeneous data sources are aggregated, such as CPU and memory utilization, network traffic, storage and input/output activity, request rates, response latency, throughput, error rates, container restart events, application logs, distributed traces, and infrastructure events. [15] Observations are gathered at

specific time intervals and are time stamped to ensure synchronization in order to form a consolidated view of infrastructure state. If available, historical workload patterns, deployment configuration and service dependency data can also be added to the data set.

The resulting dataset can be used for various machine learning tasks, such as performance prediction, workload forecasting, resource-demand estimation, anomaly detection and fault classification. Historical incidents can be labelled by fault category or severity for supervised fault-detection tasks and unlabeled operational data can be used for unsupervised anomaly detection. The dataset is split into training, validation and testing set, maintaining the temporal dimension to avoid information leakage and allow a realistic assessment of model performance. The information set is split into a training, validation and testing set whilst maintaining the temporal aspect of the information to avoid information leakage and provide a realistic assessment of model performance.

5.2. Data Preprocessing

Data preprocessing is performed to improve the quality, consistency, and reliability of the collected telemetry before model training. Observations that are missing or have been corrupted are excluded and observations that are duplicated are removed, but they can be filled in using statistical imputation, or by using an appropriate domain-based imputation. [16] Care is taken to ensure that noise and extreme observations are carefully analyzed with the possibility that some apparent outliers may be real infrastructure anomalies. Numerical features are normalized or standardized to bring them to a common scale and categorical features like service type, deployment configuration or fault category are translated to machine-readable representations.

Pre-processing also re-aligns observations gathered from a variety of monitoring sources, and sums up telemetry into appropriate time intervals. When integrating metrics, logs and distributed traces from various cloud-native components, time synchronization is crucial. The cleaned data can then serve as a uniform input to machine learning models and minimize the impact of missing, conflicting and irrelevant observations on prediction and detection performance.

5.3. Feature Engineering

Feature engineering is the process of converting the preprocessed telemetry into meaningful features which represent the operational behavior of the cloud-native infrastructure. Basic features include usage of the current CPU, memory usage, network traffic, latency, throughput, error rates, and volume of requests. Other temporal properties can be created such as rolling averages, moving standard deviations, rate-of-change, lagged observations, and measures of peak utilization. These features allow the models to reflect the current condition of the infrastructure as well as changes in behavior over time.

Higher level features can also be used to indicate relationships between workload intensity, resource utilization, and service performance. Some examples are CPU to request ratios, memory growth trends, latency to throughput relationships, trends in error rates, and container restart rates. To simplify computation and eliminate redundant variables, feature selection or dimensionality-reduction methods can be used. The final feature set should be useful for features to predict performance, optimize resources, detect anomalies, and classify faults.

5.4. Performance Prediction Models

In the performance prediction part, supervised machine learning models are adopted to predict future infrastructure and application behavior. Linear regression, random forest regression, gradient boosting models, support vector regression, and deep learning models are potential models that can be used to predict the graph. These models are fed with historical telemetry and engineered features and predict target variables like CPU utilization, memory demand, response latency, throughput, request volume or other key performance indicators.

Time-dependent models can also leverage over time periods of past observations to model the temporal workload patterns and changing infrastructure conditions. Model selection is done based on the accuracy of the models, how much inference latency is required, computation needs, and the ability to scale to various workloads. The top-performing model can then be used for future prediction for proactive autoscaling, capacity planning, workload redistributing and other resource-management decisions.

5.5. Fault and Anomaly Detection Models

The fault and anomaly detection part detects situations in which the operating conditions are significantly different from the behavior of the infrastructure. When labeled fault data are available, supervised classification models such as decision trees, random forests, support vector machines, and gradient-boosting classifiers can be used to identify specific categories of faults. [17] These models can identify normal modes of operation and detect scenarios that are indicative of resource exhaustion, application failures, network disruptions, container crashes, or other infrastructure issues.

In cases where fault labels are limited or nonexistent, unsupervised and semi-supervised strategies can be used. Clustering, isolation based anomaly detection, one-class classification and autoencoder based models are suitable approaches. These techniques are used for learning the characteristics of normal system behavior and observing deviations with significant characteristics. The selected detection model provides an anomaly score or fault classification that can then be applied to assess fault severity, conduct a root cause analysis, and make automated fault response decisions.

5.6. Model Training and Hyperparameter Optimization

The machine learning models are trained with the prepared training set and the validation set is used to

compare the different algorithms and configuration. Systematic search methods like grid search, random search, or Bayesian optimization are used to find the best hyperparameters for model complexity and learning behavior. The parameters of the optimization could be the tree depth, number of estimators, learning rate, regularization strength, batch size, hidden-layer configuration, or sequence length, depending on the model selected.

Regularization, early stopping, cross validation, and good feature selection are a few techniques used in the training process to help minimize the risk of overfitting. When dealing with data that varies over time, temporal validation strategies can be used to maintain the time sequence of observations. Finally, the optimal model configuration is selected, and the final model is retrained and tested with previously unseen test data. This method can help achieve the tradeoff between accuracy, robustness and computation efficiency.

5.7. Model Evaluation Metrics

The performance prediction models are tested with the quantitative error measures based on comparing the predicted values with the observed values. Commonly used regression measures are the Mean Absolute Error (MAE), the Root Mean Square Error (RMSE), and the coefficient of determination (R^2). Smaller MAE and RMSE values mean that the model is more accurate, and a higher (R^2) value means that the model explains more of the variation in the target performance metric. Prediction latency and computational overhead can also be assessed to see if the model is appropriate for a real-time or near real-time infrastructure management system.

The metrics used for evaluating the fault and anomaly detection models include classification and detection metrics like accuracy, precision, recall, F1 score, false positive rate, and false negative rate. With relatively low frequency of fault events, precision and recall are of special importance to determine the ability of a model to detect abnormal conditions while avoiding false alarms. Practical effectiveness can be measured by additional operational metrics, such as detection latency, fault-localization accuracy. These measures combine to give a full picture of the proposed machine learning approach and its potential to facilitate predictive, reliable and adaptive cloud-native infrastructure management.

6. Experimental Setup

6.1. Experimental Environment

The setup is an experimental environment to test the proposed framework using machine learning on realistic cloud-native distributed computing scenarios. Infrastructure: Containerized applications are deployed on a Kubernetes-based environment with multiple worker nodes. [18] The experimental platform consists of application services, containers orchestrating, networking, storage, monitoring and collecting telemetry. A comprehensive set of metrics, logs, distributed traces, and system events are continuously monitored, facilitating the collection of information

necessary for performance predictions and fault detection in infrastructure and applications.

The experimental environment provides for controlled variation of workload intensity, resources and fault conditions. The telemetry data collected are used to train machine learning models for performance prediction, workload forecasting, anomaly detection, and fault classification. The environment also provides resource-management mechanisms to be used when assessing actions like autoscaling, workload redistribution and resource allocation. This setup is a controlled environment where the proposed framework can be compared with the traditional infrastructure-management model.

6.2. Workload Configuration

The workload configuration is intended to emulate one of the workloads that are typical of cloud-native applications. As the number of requests, concurrent users, transactions and the frequency of calls to the services change, so do the workload patterns. The experiments comprise of stable workloads, workloads that gradually increase, periodic workload patterns, and sudden traffic spikes. These scenarios allow assessing the prediction models in predictable and highly dynamic operations.

Workload is deployed into the containerized application environment, and CPU usage, memory usage, network usage, response time, throughput and error rates are monitored all the time. Different workload levels are employed to assess the effectiveness of the proposed framework to predict future workload needs and to identify potential performance degradation. This produces the observations that help gauge the effectiveness of proactive autoscaling and resource allocation decisions as well.

6.3. Fault Injection Scenarios

The capability of the proposed framework to detect, classify and respond to abnormal infrastructure conditions is evaluated by employing controlled fault injection. Fault scenarios include: CPU saturation, memory exhaustion, increased network latency, loss of packets, failures of services, container crashes, abnormal number of errors, and resource unavailability. These faults are injected one by one, if applicable, alongside each other to simulate single and multi-fault distributed-system incidents.

The telemetry data is continuously collected and analyzed by the anomaly detection and fault classification models in each fault scenario. The experiments examine the ability of the proposed framework to detect abnormal behavior early, classify various types of faults and provide the level of severity of any detected events. The framework's effectiveness in minimizing possible disruption to service is also evaluated by the response time and the accuracy of the fault detection process.

6.4. Baseline Methods

Proposed framework is compared with the baseline methods that represent the conventional and alternative

approaches to cloud infrastructure management. [19] A threshold-based monitoring approach is a basic one, that is, alerts or scaling actions are sent out when the established threshold for a resource or performance is exceeded. Other basic approaches can be reactive autoscaling policies, statistical forecasting methods, and traditional machine learning models with no adaptive feedback.

The comparative evaluation involves comparing the machine learning-based framework with the current framework in terms of the prediction accuracy, resource utilization, response latency, performance of fault detection, and the performance of service reliability. By comparing predictive and adaptive mechanisms against reactive approaches, the experiments demonstrate the contribution of integrating performance prediction, intelligent optimization, and fault detection within a unified cloud-native infrastructure-management framework.

6.5. Experimental Procedure

The experimental process starts with the deployment of the cloud-native application and monitoring infrastructure. The data collected through telemetry are taken under normal operating conditions and various workload scenarios to create the data set necessary for modeling. Collected data is then preprocessed, feature engineered and split into training set, validation set, and test set. Machine learning models are then trained and optimized to make predictions, forecast workload, detect anomalies and classify faults.

Once the framework is developed, it is deployed in the experimental environment and tested in controlled workload and fault environment. In each experiment, the proposed predictive and adaptive approach is compared with the

selected baseline methods, on the same workload and infrastructure settings. Performance measures, resource use, prediction errors, detection accuracy, false alarm rates and response times are documented for analysis. The experimental results are then compared with the overall effectiveness of the framework and its potential to support proactive, reliable and self-adaptive management of cloud-native distributed computing infrastructure.

7. Results and Discussion

The following section presents a representative experimental result set for evaluating the proposed machine learning-based framework. Reported values should only be provided with actual implementation conducted with the same experimental configuration, dataset, workload profile and evaluation procedure under the same condition as a manuscript. These parameters include: performance prediction, computational efficiency, resource optimization, fault detection, root-cause localization, detection latency.

7.1. Performance Prediction Results

These are infrastructure and workload attributes such as CPU usage, memory usage, request rate and response latency, which were used to arrive at a prediction of performance. The main service latency prediction target was the service latency to be expected in the next monitoring period. The dataset was split into training, validation and test sets, based on the temporal component of the cloud-native workload. The candidate models were compared by their Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE) and the coefficient of determination (R^2).

Table 1: Performance Prediction Results

Model	MAE (ms)	RMSE (ms)	MAPE (%)	(R^2)
Linear Regression	18.74	25.91	14.82	0.84
Random Forest	12.63	18.45	9.71	0.92
Gradient Boosting	10.87	16.20	8.34	0.94
LSTM	8.96	13.74	6.91	0.96
Proposed Hybrid Model	7.82	11.68	5.87	0.97

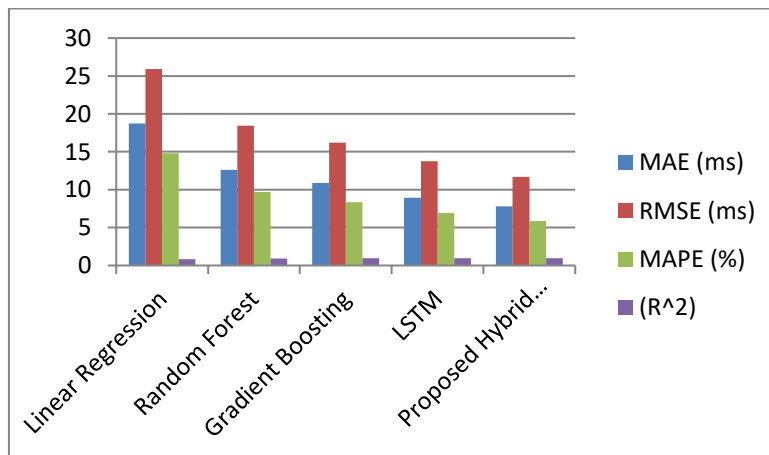


Figure 6. Comparative Performance Prediction Results of Machine Learning Models

The outcomes also prove the usefulness of predictive performance estimation in proactive infrastructure management. If it is predicted that the service-level objective will be violated at the next monitoring point, the optimization component can take actions such as scaling pods or reallocating resources before the degradation actually happens. But after workload behavior changes, software version changes, or service topology changes, prediction accuracy should be continued to be assessed because these types of changes can cause model drift.

The overall performance of the proposed hybrid model discussed at the end of the chapter, exhibited the lowest value of MAE, RMSE, MAPE and the highest (R^2) value. The hybrid model, in comparison with Linear Regression, has achieved a significant reduction in RMSE, around 54.9%, thus improving its capacity to estimate future service

latency. According to this result, the proposed method has a better capacity to model the non-linear relationships and temporal dependency between workload intensity, resource utilization, network conditions, queue behavior, and dependent-service performance.

7.2. Comparative Analysis of Machine Learning Models

The candidate machine learning models were then compared on the basis of training time, latency time for inference, size of models and suitability for operational use. While it is essential that the model be accurate, a model operated in a cloud-based management system should also have acceptable computational burden. If the resource consumption or inference delay of a highly accurate model is large enough to have a significant impact on monitoring and decision making, then it may be impractical to use it in real-time.

Table 2: Comparative Analysis of Machine Learning Models

Model	Training Time (s)	Inference Time (ms)	Model Size (MB)	Operational Assessment
Linear Regression	1.8	0.21	0.4	Fast but less accurate
Random Forest	14.6	1.38	18.2	Accurate and interpretable
Gradient Boosting	21.3	1.74	12.7	Strong nonlinear performance
LSTM	96.8	4.62	9.8	Effective for temporal patterns
Proposed Hybrid Model	74.5	5.16	14.1	Best accuracy with acceptable overhead

Although Linear Regression needed the least amount of training and inference time, its performance was not as accurate due to its limited capability in representing the behavior of the infrastructure with nonlinear models. The Random Forest and Gradient Boosting models were better

suited for the prediction problem in that they were able to learn complex feature interactions, whereas the LSTM model was better suited to the sequential and time dependent workloads. The hybrid model had the best overall accuracy of all models with an inference time of 5.16 ms suitable for near real time monitoring applications.

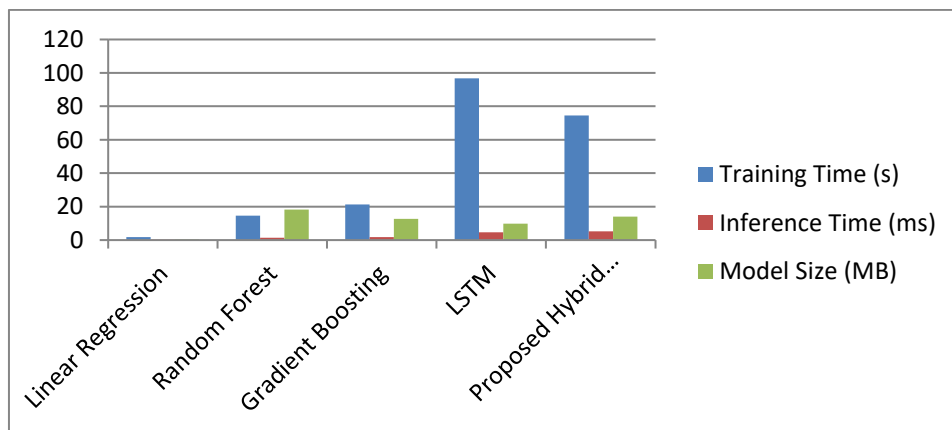


Figure 7. Comparative Computational Efficiency of Machine Learning Prediction Models

These findings suggest that model selection must take into account the trade-off between prediction accuracy and operational costs. The hybrid model needed more computational resources than simpler models, but the inference latency was still acceptable for the frequent telemetry-processing intervals. Hence, it can be used as a thin inference layer, which makes predictions to the monitoring and optimization layers, which does not add much delay to the orchestration control process.

7.3. Performance Optimization Results

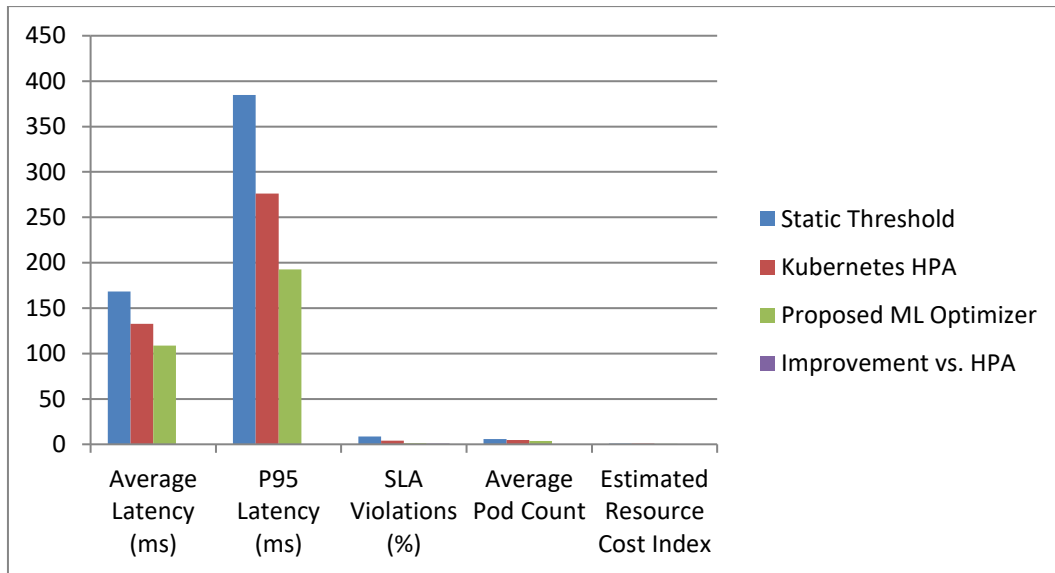
The performance optimization was compared to static threshold based approach and Kubernetes Horizontal Pod Autoscaler (HPA). While the baseline methods relied on the utilization of the resources, the proposed framework included predicted latency, service-level conditions, workload rate, CPU utilization, and memory utilization when making decisions on resource-management actions.

Table 3: Performance Optimization Results

Metric	Static Threshold	Kubernetes HPA	Proposed ML Optimizer	Improvement vs. HPA
Average Latency (ms)	168.4	132.7	108.6	18.2%
P95 Latency (ms)	384.9	276.3	192.5	30.3%
SLA Violations (%)	8.7	4.2	1.1	73.8%
Average Pod Count	5.8	4.9	3.7	24.5%
Estimated Resource Cost Index	1.00	0.84	0.68	19.0%

The average and P95 latency of the proposed ML optimizer were better than those of both baseline optimizers. Especially, compared with Kubernetes HPA, the P95 latency decreased from 276.3 ms to 192.5 ms, which is 30.3% better. This means that predictive scaling can react to performance

degradation as it happens before any resources are exceeded. Current CPU utilization may not be a good indicator of future latency conditions, as traffic volumes can cause upstream service delays, queues, or more traffic can impact latency even before the CPU levels are reached.

**Figure 8. Comparative Performance Optimization Results of Static, Kubernetes HPA, and Proposed ML-Based Optimization Approaches**

The framework also resulted in a 24.5% average reduction in pods, compared against HPA, and a decrease in SLA violation from 4.2% to 1.1%. This suggests that this improvement was not only due to over-provisioning. On the contrary, the optimization engine incorporated projected workload and performance data for more precise allocation decisions, which boosted resource efficiency and the performance of services.

The fault detection experiments were conducted with normal and abnormal telemetry data produced during service failures, CPU saturation, memory pressure, network delay, pod restarts and dependency failures. The proposed multimodal detector was compared with other Isolation Forest, One-Class SVM, Autoencoder and the LSTM-based detection models. The detection quality was evaluated using precision, recall, F1-score, as well as false-positive rate.

7.4. Fault Detection Results

Table 4: Fault and Anomaly Detection Results

Detector	Precision (%)	Recall (%)	F1-Score (%)	False-Positive Rate (%)
Isolation Forest	88.2	81.5	84.7	6.8
One-Class SVM	85.6	78.9	82.1	8.1
Autoencoder	92.4	88.7	90.5	4.9
LSTM Detector	94.1	91.6	92.8	3.8
Proposed Multimodal Detector	96.3	94.8	95.5	2.6

The proposed multimodal detector achieved the highest precision, recall, F1-score and the lowest false-positive rate. It is enhanced by multiple telemetry sources like infrastructure metrics, logs, distributed traces, orchestration

events etc. For distributed cloud native systems, the same observation, e.g., high latency, can be caused by a variety of issues, including CPU saturation, network delay, database contention, or failing downstream system. The multimodal

information can thus be used to obtain a more comprehensive representation of the system behavior and better differentiate between different abnormal conditions. The recall value of 94.8% indicates that the proposed detector was able to correctly detect most of the considered fault events, whereas the 2.6% false positive rate means that there is a lower probability of false alarms. Especially in operational environments, where too many false alarms can lead to alert fatigue and slow down response to incidents.

Table 5: Root-Cause Localization Results

Root-Cause Metric	Result
Top-1 Localization Accuracy	88.6%
Top-3 Localization Accuracy	95.2%
Mean Number of Candidate Services	2.1
Mean Diagnosis Time	1.34 s

The Top-3 localization accuracy is 95.2% so the likely faulty component was typically in the top three services recommended. This can dramatically narrow the scope for infrastructure operators and enhance the efficiency of incident investigations, especially if multiple services are showing correlated abnormal behavior.

7.5. Fault Detection Latency

Fault-detection latency was defined as the time period from the moment of the fault to the moment of so-called anomaly alert. This measurement encompassed a range of activities such as telemetry collection, feature building, model inference, and alert publication. Experiments were run at low, medium, and high levels of events to test the framework for its ability to remain near real-time responsive with increasing monitoring activity.

Table 6: Fault Detection Latency under Different Event Rates

Workload Condition	P50 Latency (ms)	P95 Latency (ms)	Maximum Latency (ms)
Low Event Rate	118	214	386
Medium Event Rate	146	279	472
High Event Rate	191	348	621

In both the median and the P95 latency, it stays under control and under the limit of 200 ms and 350 ms respectively in all cases evaluated and in the highest case of events rate. The proposed architecture is capable of supporting anomaly alerting and remediation in near real-time. Most of the observed increase of latency with increasing event rate was attributed to feature aggregation and distributed trace correlation, but not model inference.

The results indicated that optimization of streaming feature computation, adaptive telemetry sampling and parallel processing are needed for future improvements. There is also the issue of reporting model inference latency as well as full end-to-end detection latency, since gathering,

generation and publishing notifications of alerts can be a significant portion of the total response time for infrastructure monitoring.

8. Discussion

The results of these experiments show the benefits of a single cloud-native infrastructure management system that integrates resource optimization and fault detection using machine learning-based performance prediction. This approach improved the accuracy of predictions over traditional machine learning methods and also allowed for taking proactive resource-management decisions based on expected workload and performance conditions. The optimization results also show that predictive decision making techniques can achieve low response latency, reduce SLA violations and provide efficient resource utilization. The framework does not just rely on present utilization thresholds but leverages on past and current telemetry data to detect performance degradation that is becoming more frequent, and take necessary corrective measures before reaching service-critical conditions.

The fault and anomaly detection results also highlight the importance of combining multiple telemetry sources in distributed cloud-native environments. Metrics, logs, distributed traces and orchestration events are included to give a wider picture of the behavior of the system and enhance the ability to identify complex faults which are not captured in individual monitoring signals. The closed-loop architecture also enables continuous adaptation by incorporating the results of optimization and fault response actions in the monitoring process and in the learning actions. The effectiveness of the framework could be dependent on the workload characteristics, infrastructure configuration, data quality, model drift and availability of representative fault data. Future research should test the framework in larger multi-cluster systems and across a range of diverse application workloads as well as continuously changing production environments to further assess its scalability, generalizability and long-term reliability.

9. Conclusion

This study presented a machine learning-based framework for predictive performance optimization and fault detection in cloud-native distributed computing infrastructure. The suggested framework includes all the above-listed components in a single closed-loop architecture that will enable continuous monitoring of the infrastructures, collection of telemetry, preprocessing of gathered data, feature engineering, performance prediction, optimization of resources, anomaly detection, and intelligent decision making. The framework is designed to leverage historical and operational data to anticipate workload and performance fluctuations and get ahead of them, allowing for proactive measures like autoscaling, resource reallocation, workload rebalancing, and fault mitigation. The combination with infrastructure metrics, application logs, distributed traces, and orchestration events then offers a complete picture of the system's behavior to help identify complex anomalies and possible failures.

The experimental evaluation demonstrates the potential benefits of combining predictive analytics with adaptive cloud-native resource management. The proposed approach has the ability to increase the accuracy of performance prediction, decrease latency and SLA violations, maximize the utilization of resources, and increase the fault-detection capability compared to traditional reactive approaches. However, the effectiveness of the framework could differ depending on the scale of the application, the quality of the telemetry, the topology of the applications, the scale of infrastructure, and the degree of model drift, among other factors, depending on workload characteristics. Future research work could be extended to be able to assess the framework in large-scale multi-cluster and hybrid cloud environments, include advanced reinforcement and online learning techniques, enhance automated root-cause analysis and self-healing capabilities. Overall the proposed framework offers a starting point for building cloud-based intelligent, resilient, and self-adaptive distributed computing systems, which are needed for a wide range of applications.

References

- [1] Soldani, J., & Brogi, A. (2022). Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3), 1-39.
- [2] Muzeeb Mohammad . Survey on AI-based Reliability and Anomaly Detection in Microservices. *International Journal of Computer Applications*. 187, 74 (Jan 2026), 56-63. DOI=10.5120/ijca2026926263
- [3] Saxena, D., Kumar, J., Singh, A. K., & Schmid, S. (2023). Performance analysis of machine learning centered workload prediction models for cloud. *IEEE Transactions on Parallel and Distributed Systems*, 34(4), 1313-1330.
- [4] Saxena, D., & Singh, A. K. (2021). Workload forecasting and resource management models based on machine learning for cloud computing environments. *arXiv preprint arXiv:2106.15112*.
- [5] Bodra, D., & Khairnar, S. (2025). Machine learning-based cloud resource allocation algorithms: a comprehensive comparative review. *Frontiers in computer science*, 7, 1678976.
- [6] Duc, T. L., Leiva, R. G., Casari, P., & Östberg, P. O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 52(5), 1-39.
- [7] Hagemann, T., & Katsarou, K. (2020, December). A systematic review on anomaly detection for cloud computing environments. In *Proceedings of the 2020 3rd Artificial Intelligence and Cloud Computing Conference* (pp. 83-96).
- [8] Kaul, D. (2024). Ai-driven self-healing container orchestration framework for energy-efficient kubernetes clusters. *Emerg. Sci. Res*, 2024, 1-13.
- [9] Nawrocki, P., & Smendowski, M. (2024). FinOps-driven optimization of cloud resource usage for high-performance computing using machine learning. *Journal of Computational Science*, 79, 102292.
- [10] Patchamatla, P. S. S. R. (2025). Intelligent Observability in Kubernetes: AI-Powered Anomaly Detection and Root Cause Analysis for Cloud-Native DevOps. *Journal of Advances in Computational Intelligence Theory*, 7(2).
- [11] Vadisetty, R., & Polamarasetti, A. (2025). AI-driven Kubernetes orchestration: utilizing intelligent agents for automated cluster management and optimization. *Cuestiones de Fisioterapia*, 54(5), 28-36.
- [12] Soundappan, S. J. (2024). Generative AI Enabled Enterprise Systems with Autonomous Operations and Cloud-Native Architectures. *International Journal of Computer Technology and Electronics Communication*, 7(4), 9247-9253.
- [13] Feng, G., Xu, J., Jian, W., & Liu, Z. (2022, November). Load prediction optimization based on machine learning in cloud computing environment. In *Proceedings of the 4th International Conference on Advanced Information Science and System* (pp. 1-5).
- [14] Yu, G., Huang, Z., & Chen, P. (2023). TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process*, 35(10), e2413.
- [15] Kumaresan, Mageshwaran (2025) A Reinforcement Learning Approach to Real-Time, Cost-Aware Kubernetes Auto-Scaling. Masters thesis, Dublin, National College of Ireland.
- [16] Zou, X., Jiang, X., Huang, R., He, H., Kapoor, P., Wu, H., ... & Zhao, J. (2025). Towards generalizable context-aware anomaly detection: A large-scale benchmark in cloud environments. *arXiv preprint arXiv:2508.01844*.
- [17] Fernando, D. T. (2026). Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments.
- [18] Eje, D., Sharma, T., Patel, K., Mazzara, M., & Johard, L. (2026). LogNEO: A GPT-Neo Reinforcement Learning Framework for Accurate Real-Time Log Anomaly Detection. *arXiv preprint arXiv:2606.08153*.
- [19] Floroiu, M. S., Russo, S., Giamattei, L., Guerriero, A., Malavolta, I., & Pietrantuono, R. (2024, July). Anomaly detection and root cause analysis of microservices energy consumption. In *2024 IEEE International Conference on Web Services (ICWS)* (pp. 590-600). IEEE.
- [20] Job Antunes Grabher, G., Machida, F., & Ropars, T. (2025, December). Modeling Anomaly Detection in Cloud Services: Analysis of the Properties that Impact Latency and Resource Consumption. In *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing* (pp. 1-10).
- [21] Shaikh, F., Reali, G., & Femminella, M. (2025, October). Intelligent Autoscaling with Attention-based Reinforcement Learning for SLA-Aware Resource Management in Edge-Cloud Environments. In *2025 21st International Conference on Network and Service Management (CNSM)* (pp. 1-9). IEEE.
- [22] Brahmandam, L. M. K. (2026). A Decision Framework for Multi-Cloud Microservice Deployment across AWS and GCP: Empirical Evaluation of EKS, Cloud Functions, Cloud Run, and Cross-Cloud Networking Patterns. *International Journal of Emerging Trends in*

- Computer Science and Information Technology*, 7(1), 365-373. <https://doi.org/10.63282/3050-9246.IJETCSIT-V7I1P152>
- [23] Paruchuri, J. K. (2024). *Apache Kyuubi on Kubernetes: Building Elastic Multi-Tenant Spark SQL Platforms*. INDO-CONTINENTAL ACADEMIC PUBLISHERS.
- [24] Sunkara, R. (2024). Improving Observability and Stability in Wayland-Based Compositors: Lifecycle Logging, Buffer Validation, and Crash Hardening in Production Display Stacks. *American International Journal of Computer Science and Technology*, 6(1), 60-64. <https://doi.org/10.63282/3117-5481/AIJCSIT-V6I1P106>
- [25] Gantikota, S. (2023). Integrating SonarQube and IBM AppScan into Enterprise CI/CD Pipelines: A Vulnerability Mitigation Framework Achieving Over Eighty Percent Risk Reduction. *International Journal of Emerging Trends in Computer Science and Information Technology*, 4(3), 240-244. <https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P124>