A Predictive Auto-Scaling for Cloud Application

Master Thesis

by

Chaithra Jagannatha Rao Telkar

Matriculation number: 11037491

2025-10-27

SRH University Heidelberg

School of Information, Media and Design

Master of Applied Computer Science (ACS)

Major field "UI/UX Development"

First Supervisor

Prof. Dr. Gerd Moeckel

Second Supervisor

Paul Tanzer

Declaration of Authorships

I hereby declare that my herewith submitted paper is my own original work. I have written it independently without outside help and have not used any sources other than those indicated

- in particular, no sources not named in the references.

I have appropriately indicated any direct quotations or passages taken from literature, and the use of intellectual property from other authors, by providing the necessary citations within the work. This applies equally to the sources used for text generation by Artificial Intelligence (AI).

I hereby declare that the paper was not previously presented to another examination board, and I also confirm that the PDF version of this paper is exactly identical in content to the hard copy.

Heidelberg, Germany (place) 27-10-2025 (date)

Chaithra Jagannatha Rao Telkar (signature)

Abstract

Chaithra Jagannatha Rao Telkar, School of Informatics,

SRH University Heidelberg Master Thesis

Predictive Auto-Scaling for Cloud Application

The project presenting the predictive auto-scaling framework which combines machine learning models with real-time cloud monitoring for the proactive resource management. The system utilizes the three predictive models such as Random Forest, LSTM, and ARIMA to forecast workload trends and the response time variations from the historical performance data. A custom-built controller interprets these predictive signals that applies uncertainty margins, and it initiates scaling actions before any potential performance degradation occurs. The Prometheus and Grafana where integrated to provide complete observability, offering the real-time visualization and the traceability from the model predictions to scaling executions. Overall, the framework achieves improved system stability, minimizes in latency violations, and enhances the resource utilization efficiency compared to the conventional reactive scaling mechanisms.

Table of Contents

1 Introduction 1

1.1 **Background of Cloud Computing** 1

1.2 **Research Problem** 4

1.3 **Research Objectives** 5

1.4 **Motivation** 7

1.5 **Research Scope** 7

1.6 **Organization of the Thesis** 8

2 Literature Review 10

2.1 **Introduction to Cloud Auto-Scaling** 10

2.2 **Traditional Auto-Scaling Techniques** 10

2.3 **Predictive Auto-Scaling Approaches** 12

2.3.1 Statistical Forecasting Models 12

2.3.2 Approaches that are based on machine learning 12

2.3.3 Time- series forecasting with Deep Learning models 12

2.3.4 Meta-Learning and Re-enforcement Learning 13

2.4 **Feature Engineering and Workload Prediction** 13

2.4.1 Workload Trace and Telemetry Data Significance 13

2.4.2 Time-Series Decomposition 14

2.4.3 Lag Characteristics and Past Reliances 14

2.4.4 Analysis of Metrics Correlation 14

2.4.5 Dealing with an anomaly and outliers 14

2.5 **Comparative Studies of Predictive Models** 15

2.5.1 Classical Statistical Models vs. Machine Learning 15

2.5.2 Deep Learning Approaches 15

2.5.3 Hybrid and Ensemble Models 16

2.5.4 Benchmarking against Reactive Methods 16

2.6 **Hybrid and Advanced Scaling Strategies** 17

2.6.1 Statistical-Machine Learning Hybrid Models 17

2.6.2 Deep Learning Ensembles.....17
2.6.3 Auto-Scaling based on reinforcement learning.....17
2.6.4 Evolutionary and Meta Heuristic Optimization.....18
2.6.5 Cloud-Native Systems Azure AI Orchestration.....18
2.7 Research Gaps in Current Literature.....18
2.7.1 Lack of Flexibility to Rapid workload increases.....18
2.7.2 The lack of integration with commercial cloud platforms.....18
2.7.3 Energy and Sustainability Concerns.....19
2.7.4 Low Multi-Cloud and Edge Support.....19
2.7.5 Lack of Explainability in Predictive Models.....19
3 System Design and Methodology.....22
3.1 Introduction.....22
3.2 System Architecture.....23
3.2.1 Input Layer.....24
3.2.2 Processing Layer.....26
3.2.3 Decision Layer.....28
3.2.4 Feedback and Monitoring.....30
3.3 Dataset Description.....32
3.4 Data Preprocessing.....34
3.5 Feature Engineering.....36
3.6 Predictive Modeling.....38
3.6.1 Long Short-Term Memory (LSTM) Networks.....38
3.6.2 Random Forest Regression.....39
3.6.3 Auto-Regressive Integrated Moving Average(ARIMA).....39
3.6.4 Baseline: Reactive Auto-Scaling.....40
3.7 Evaluation Metrics.....41
3.8 Experimental Setup.....43
4 Results and Evaluation.....45
4.1 Model Performance.....46
4.2 Predictive Signal & Decisioning.....47

4.2.1 Prometheus Monitoring and Alert Rules **48**
4.2.2 Prometheus alerts for system **49**
4.2.3 Execution of Predictive Signal Generation Pipeline **50**
4.2.4 Verification of Core Service Deployment and Status **51**
4.2.5 Predictive Upper vs Threshold **53**
4.2.6 Wants-Scale Signal and Replica Adjustment Relationship **55**
4.2.7 Recent Scaling Events and Decision Counter Analysis **57**
4.2.8 Controller Log Excerpt Demonstrating Predictive Scaling Execution **59**
4.2.9 Prometheus-Exposed Metrics of the Predictive Auto-Scaling Controller. **60**
4.2.10 LSTM Workload Prediction **62**
4.2.11 Random Forest Workload Prediction **63**

4.3 Discussion **65**

5 Conclusion and Future Work ***68***

5.1 Conclusion **68**

5.2 Limitations **69**

5.3 Future work **70**

6 References ***71***

1. Introduction

1.1 Background of Cloud Computing

Cloud computing has transformed the manner in which the computational resources are provided, consumed and managed through offering on-demand, scalable and elastic services through the Internet. Cloud computing has a pay-as-you-go billing model unlike the traditional IT infrastructures where organizations spend considerably in buying of physical servers and maintenance; therefore, cloud computing allows organizations to save on initial costs as well as enhance efficiency in their operations. The three fundamental service models, Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a Service (SaaS) can provide different degrees of abstraction, depending on the needs of the business, and the deployment models of the service include public, private, hybrid, and multi-cloud environments that provide organizations with a wide range of options [1].

Industry reports in the recent times show that cloud computing has attained ubiquitous usage in the industry in fields such as health, e-commerce, finance, education, and scientific research. In a study by Gartner, it is said that over 94% of the businesses across the world use at least one cloud service and, the cloud infrastructure investment will exceed USD 1 trillion by the year 2030. The rapid growth in the cloud adoption is mainly driven by increasing demand for the data-intensive applications, real-time analytics, artificial intelligence (AI), and Internet of Things (IoT) environments that requires high computational power and the minimal latency. However, in spite of having these advantages, cloud computing continues to face critical challenges in the resource management and security. One of the major challenges is auto-scaling, which ensures that the computing resources dynamically adjust to the workload fluctuations. Traditional reactive auto-scaling mechanisms often fail to adapt effectively to the real-time demands, resulting in SLA violations, under-provisioning (leading to performance degradation), or over-provisioning (causing unnecessary resource wastage and the cost overheads). To direct these limitations, predictive auto-scaling driven by machine learning techniques has emerged as a more intelligent solutions, capable of forecasting workload variations and enabling proactive scaling before the demands spikes occur. Another important challenge in the cloud environments is network traffic control and classification, which plays the vital role in ensuring system security, maintaining

Quality of Service (QoS), and achieving efficient bandwidth utilization. Traditional methods that rely on port number or payload signature are becoming less effective with encrypted traffic, emerging cyber threats, and bad manners of adversaries in the cloud infrastructures. In contrast, machine learning and deep learning models have demonstrated superior accuracy in the classification of network flows, intrusion detection, and the identification of anomalous patterns within extensive traffic data sets.[33] Considering these issues, the researchers and cloud service providers (CSPs) now pay attention to the development of intelligent, self-adaptive, and secure cloud systems. Resource allocation predictiveness and traffic-sensitive classification significantly increase cost-efficiency, scalability, and cyber resilience to attacks, including DDoS, ransomware, and data exfiltration [4].

Figure 1.1 shows that IaaS gives the basic infrastructure, PaaS gives platforms for making applications, and SaaS gives end users applications that are ready to use. The following two challenges and opportunities drive the current thesis, which is a study on two problems that are intertwined:

1. Auto-scaling in cloud-based apps that is based on machine learning.
2. Cloud environment Network traffic classification with advanced classification techniques.

The solutions to these two issues make this research significant in the creation of next-generation cloud ecosystems that can provide reliable, cost effective and secure services to the end-users under dynamic conditions [5].

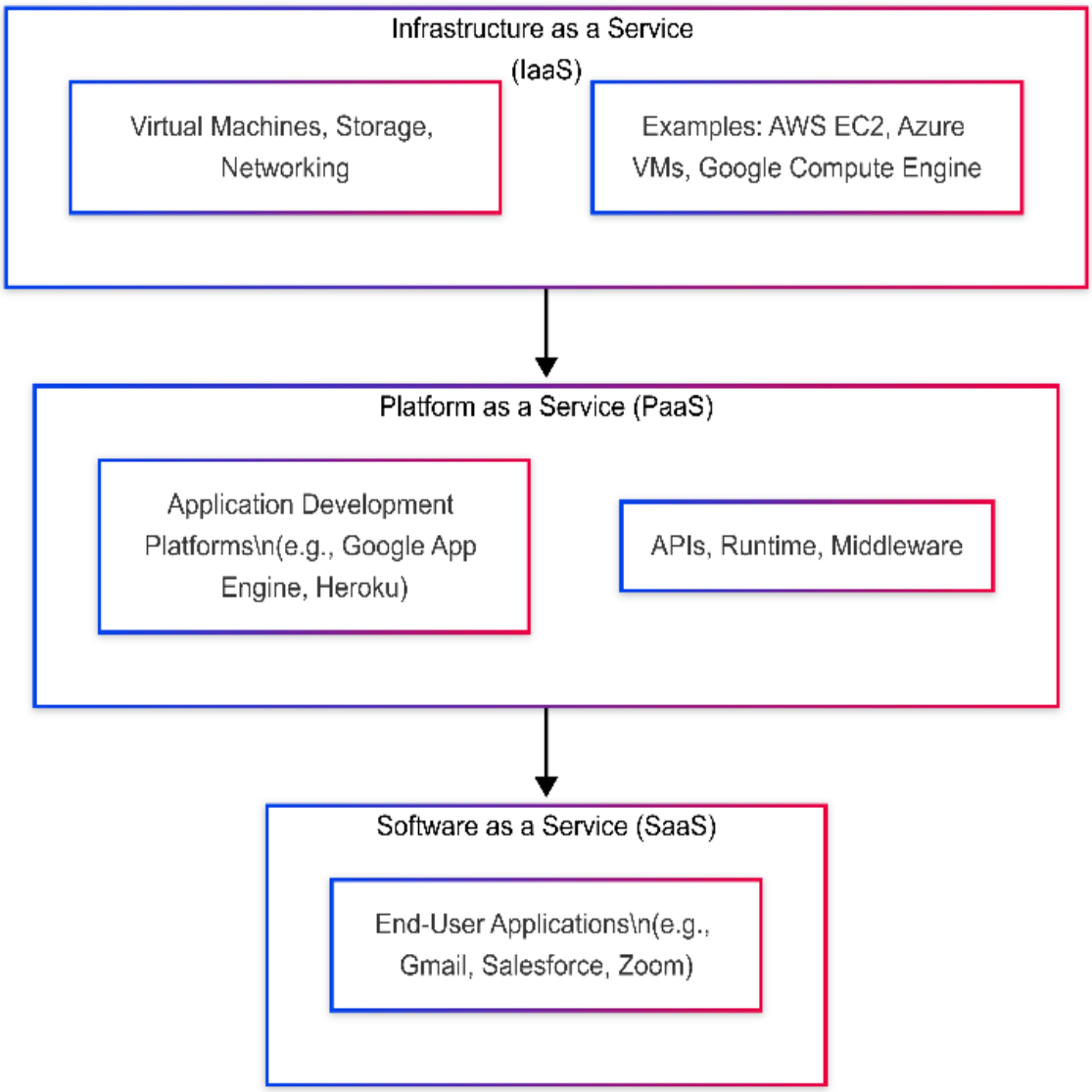

Fig 1.1 Cloud Computing Service Models

1.2 Research Problem

Despite significant advances in cloud resource management, existing auto-scaling techniques remain predominantly reactive in nature. Static thresholds, such as CPU or memory utilisation exceeding 80%, are used by reactive strategies to initiate scaling decisions. Although these methods work well for light workloads, they have three main drawbacks:

1. Latency in Scaling: Reactive policies take action only after observing spikes or declines in demand, which causes them to react slowly. SLA violations, the worse user experience, and possible system outages during periods of high demand can all resulting from this latency.
2. Ineffective Resource Utilisation: The Threshold-based approaches frequently resulting in either under-provisioning, where performance bottlenecks are caused by the insufficient capacity, or over-provisioning, where the resources are in idle state and incur needless operating costs [6].
3. Absence of Adaptive Forecasting: Complex, nonlinear, and bursty workloads behaviours in cloud-native applications are not taken into account by the traditional reactive models. They cannot anticipate abrupt changes in the workload or use contextual and historical data to make it proactive decisions [7].

Therefore, creating an intelligent predictive auto-scaling framework that overcomes these drawbacks is the main research challenge. Using real-time telemetry data and historical workload traces in the conjunction with machine learning techniques, such a framework must proactively scaling the resources, predict workload variations, and maintain an ideal balance between dependability, cost-effectiveness, and performance [8].

1.3 Research Objectives

The main goal of the thesis is to create and test a A Machine Learning Driven Predictive Auto-Scaling Framework Using Data Workloads of E-Commerce Applications. The specific objectives encompass:

1. To critically assess existing cloud computing auto-scaling techniques, in which identifying their short comings in the handling dynamic and the diverse workloads.
2. To collect and pre-process the real-world workload datasets (e.g., Alibaba Cluster Trace 2018, Google Borg traces, VMCloud data) by performing normalizations, encoding, the anomaly detection, and the time-series decomposition for the clean and structured analysis.
3. To engineer relevant features (e.g., lag variables, network-CPU correlations, anomaly indicators, seasonal-trend decomposition) which increase the predictive capability of the auto-scaling models.
4. To generate and apply predictive models that uses Random Forest regression for structured workload prediction and LSTM networks for time-series forecasting. These models will help in optimizing through hyperparameter tuning and evaluated against benchmark reactive methods.
5. To assess the predictive models' performance using statistical measures like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and R^2 score, along with visualizations to demonstrate improvements over traditional methods.
6. To validate the proposed framework’s effectiveness in achieving enhanced system performance, reduced SLA violations, improved cost efficiency, and proactive resource allocation in real-world cloud scenarios.

To achieve the research objectives, a predictive auto-scaling framework is proposed, which integrates workload data, machine learning models, and proactive scaling decisions. The general architecture of this framework is illustrated in Figure 1.2.

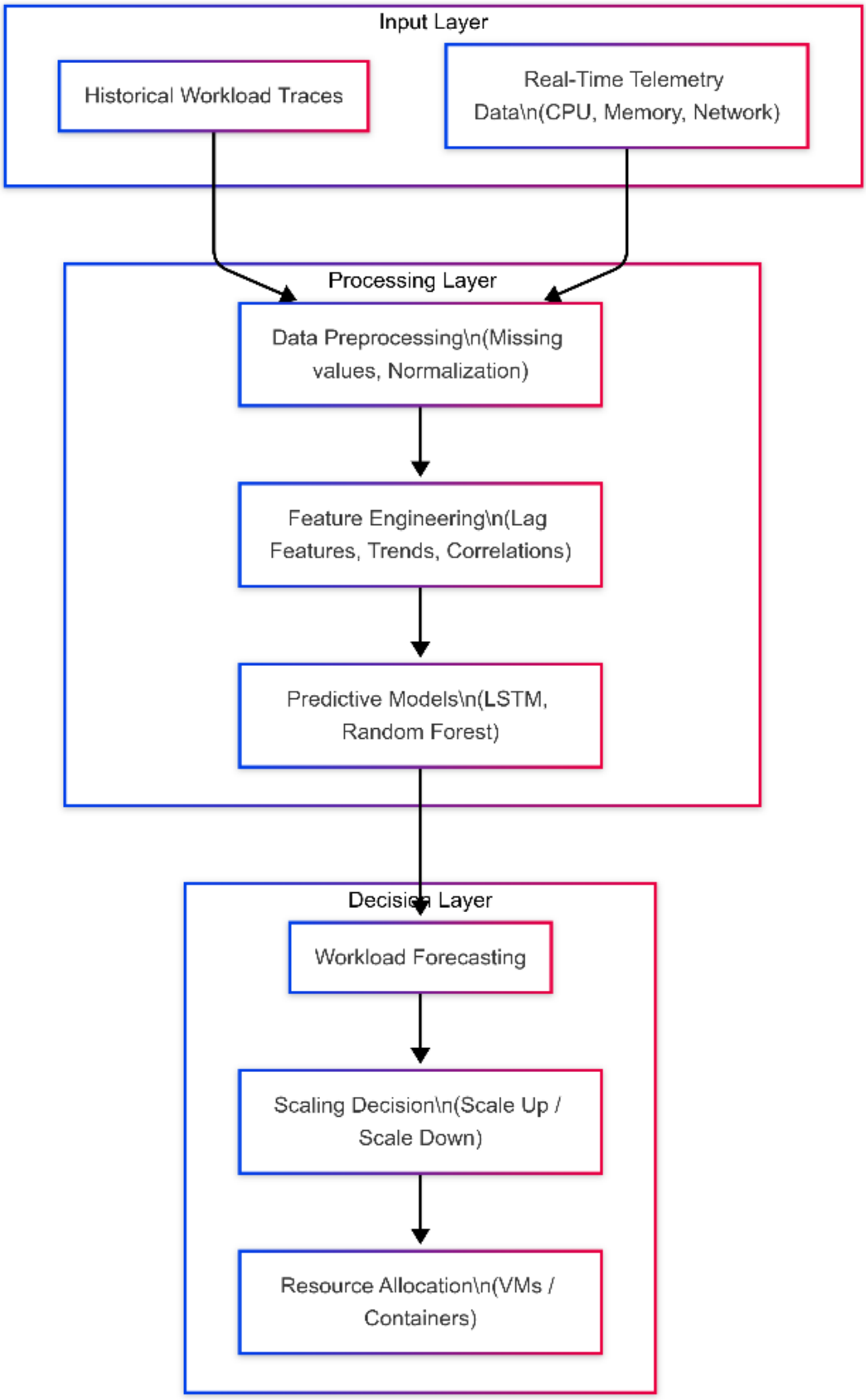

Fig.1.2 General Architecture of Predictive Auto-Scaling Framework

1.4 Motivation

Elasticity and responsiveness are now the characteristics of cloud infrastructures due to the blistering development of cloud-native applications. The thousands of users who are using applications at the same time can lose a lot of revenue and user experiences because even the slightest delays in provisioning of resources can lead to losses. An example is that massive e-commerce and streaming systems have random bursts of traffic that can hardly be managed by reactive scaling models. Equally, businesses with mission-critical workloads, like financial transactions or analytics in healthcare, cannot sustain a downtime or resource underload caused by resource bottlenecks. Additionally, the cost of cloud is a large percentage of the IT spending of organizations, and the management of the cloud resources is a business priority. Excessive provisioning leads to the wastage of an investment whereas, inadequate provisioning is likely to cause customer dissatisfaction, and SLA breaches. Predictive auto-scaling is a machine learning-driven solution that can be used to provide an active solution by using the past workload trends and predicting the demand ahead of time. This is not only more reliable but also lowers the cost of the infrastructure giving twofold advantage of optimization of performance and economic efficiency. The purpose of this study is to deal with these real world and industry-related problems and come up with an intelligent, scalable and adaptive solution to such problems to be able to work in heterogeneous and dynamic conditions.

1.5 Research Scope

This study is confined to the design and evaluation of a Machine Learning-Driven Predictive Auto-Scaling Framework Using Data Workloads of E-Commerce Applications. Its scope is outlined below:

1. Focus Areas:
    - Cloud resource allocation at the IaaS level (CPU, memory, and network utilization).
    - Forecasting of workload variations using time-series and structured prediction models.

- Integration of predictive auto-scaling with existing simulation environments (e.g., Alibaba Cluster Trace 2018, CloudSim, Google Borg traces, VMCloud datasets).

2. Exclusions:
    - The research does not develop a new cloud infrastructure from scratch but instead works within existing cloud simulation frameworks and datasets.
    - Focus is limited to performance and cost efficiency; advanced aspects such as energy-aware scaling or multi-cloud orchestration are outside the present scope.
3. Constraints:
    - Dependence on the quality and completeness of publicly available datasets.
    - Evaluation within simulated or testbed environments rather than full-scale deployment in commercial clouds.

Thus, the scope ensures that while the research remains practical and implementable, it also provides a foundation for future work in real-world and multi-cloud environments.

1.6 Organization of the Thesis

There are five chapters in this thesis:

Chapter 1: Introduction explains the history of cloud computing, the problem and goals of the research, and the purpose, extent, and general organisation of the work.

Chapter 2: Review of Literature provides a thorough analysis of cloud computing auto-scaling, machine learning-based predictive techniques, and the research gaps that drive the proposed study.

Chapter 3: Methodology and System Design explains the preprocessing and feature engineering techniques, the datasets used, the architecture of the suggested predictive auto-scaling framework, and the specifics of the machine learning models put into practice.

Chapter 4: Results and Evaluation presents the experimental outcomes of applying predictive models, assesses them using metrics like MAE, RMSE, MSE, and R2, and

contrasts their performance with more conventional reactive methods. Screenshots and graphic plots are used to show the results.

Chapter 5: Conclusion and Future Work discusses possible avenues for future development, including integrating advanced learning techniques, extending predictive scaling to multi-cloud and edge environments, and implementing the framework in actual cloud platforms. It also summarises the main conclusions and contributions of the study.

2. Literature Review

2.1 Introduction to Cloud Auto-Scaling

Cloud computing has transformed how the IT infrastructure is designed, as it provides elastic and on demand IT resources that can be dynamically scaled according to user demand. One of the enablers of this flexibility is the auto-scaling whereby computing resources including CPU, memory and bandwidth automatically scale according to the changes in the workload. The aim is to ensure performance and reliability and reduce operational expenses. Auto-scaling can be broadly divided into reactive and predictive. Reactive auto-scaling is a reaction to violations of thresholds (e.g. CPU utilization over 80 percent), where scaling actions are taken as the performance is degraded. Though useful in pure applications, reactive designs can add latency and resource wastage, especially when there is an abrupt increase in traffic. In contrast, predictive auto-scaling is based on statistical and machine learning methods to be able to predict changes in workload, which then pre-provision available resources. This minimizes the SLA violations, enhances response time, and is cost effective [9], [10].

As the number of cloud-native and latency-sensitive applications is on the upswing, predictive auto-scaling frameworks are in higher demand. The classic, stagnant techniques cannot handle a bursty and nonlinear workload, i.e., e-commerce systems, streaming systems, and IoT-based systems. Predictive scaling allows cloud providers to scale their applications and sustain scalability, economic efficiency by predicting workload changes with the help of such models as ARIMA, Random Forest, and LSTM [11], [12].

2.2 Traditional Auto-Scaling Techniques

Historical approaches to auto-scaling in cloud computing are mostly reactive, i.e. scaling responses are generated by pre-determined resource usage conditions or heuristic guidelines. The approaches have become popular in commercial cloud services (e.g., AWS Auto Scaling, Microsoft Azure Virtual Machine Scale Sets, Google Cloud Autoscaler) because they are extremely simple and easy to implement [13]. The most popular is the threshold-based scaling where resources are added or removed as the utilization measures like CPU, memory or network bandwidth go above or below predetermined thresholds. As an illustration, when the CPU utilization is above 80 percent during five consecutive minutes, a new instance of a virtual machine is started; when the utilization falls below 20

percent, there is deallocation of the resources. Although this is a simple approach, it has a number of flaws:

- Latency of Scaling Actions: The scaling actions become only performed on the realization of performance degradation which means that there will be possible breach of SLA.
- Over- and Under-Provisioning: Thresholds are usually fixed and result in over-allocation of resources when the demand is low and inadequate provisioning when there is a sudden increase in traffic.
- Absence of Adaptability: These systems fail to consider workload trends, seasonality, nonlinear dynamics, and therefore are not applicable in dynamic and uncertain settings.
- Rule-based scaling is another category of scaling that is widely used in which scaling policies are guided by multiple conditions and heuristic rules.

As an example, regulations can integrate CPU utilization and request delay or network I/O and then go on to instigate actions. Despite the flexibility of the rule-based methods as compared to the single-threshold models, they are reactive and they still do not predict trends in workloads. Other previous papers investigated heuristically-based and time-based scaling, in which the scaling activities are pre-programmed at a certain time (e.g., scaling up in business hours). Although these techniques work well in the predictable workload context, they are infeasible in the cloud-native application context where traffic is highly dynamic and frequently bursty [14]. In short, the conventional auto-scaling solutions presented a valuable base of elastic management on clouds but is still restricted by being reactive. It is these flaws that have led to predictive and intelligent auto-scaling solutions, which use machine learning and deep learning algorithms to make predictions about demand and provide resources in advance [16].

2.3 Predictive Auto-Scaling Approaches

To mitigate drawbacks of the conventional method of threshold-based and rule-based approaches, the scholars have forwarded the predictive auto-scaling strategies, which actively allocate resources, with the help of forecast of workload demand in future. The main concept is to process history workload traces, telemetry data, and contextual information and forecast variations prior to their happening, thereby guaranteeing resources allocation in time and minimized latency and costs. Of the greater number of statistical forecasting models, only the most commonly used ones will be presented here:

2.3.1 Statistical Forecasting Models

The classical statistical techniques used in early predictive methods include Autoregressive Integrated moving average (ARIMA), holt winters exponential smoothing and Prophet. They are used when the demand pattern is somewhat stable and linear trends and seasonality are required to capture the workload data. Nevertheless, they fail to perform well when nonlinear or highly bursty workloads are applied, which are prevalent in a current cloud environment [17].

2.3.2 Approaches that are based on machine learning

Machine learning (ML) methods are more flexible in responding to various datasets and nonlinear behavior. The predictive model of workload demand on the basis of CPU, memory, and network measures has been done on Support Vector Machines (SVMs), random forests, gradient boosted trees and regression models. The advantages of ML-based solutions are feature engineering (such as lag variables, anomaly indicators, correlations between traffic and utilization) and the ability to adjust to various types of workloads, which is more accurate than using purely statistical techniques.

2.3.3 Time- series forecasting with Deep Learning models

Time-series forecasting Time-series forecasting Time- series forecasting Deep learning models have demonstrated better performance than alternative forecasting methods, especially those based on artificial intelligence, by modeling long-term temporal effects on workload traces. LSTMs are, in particular, useful in the prediction of demand spikes in web applications, data analytics systems and containerized workloads. Convolutional

Neural Networks (CNNs) are also used to identify workload pattern in recent research and show excellent performance when used in hybrid CNN-RNN models.

2.3.4 Meta-Learning and Re-enforcement Learning

New studies study the reinforcement learning (RL) and meta-learning to auto-scale. The agents of RL learn the best scaling policies by trial-and-error in the environments provided by clouds, which is performance and cost-balanced. Instead, meta-learning strategies are designed to predict the behavior of new workload types with predictive models and therefore enable them to scale adaptively when operating on a heterogeneous cloud. These methods, yet continuously developing, point to the way of autonomous and smart cloud management systems [18]. To conclude, predictive auto-scaling can be based on a broad variety of methods, including classical time-series methods of forecasting as well as more advanced deep learning-based or reinforcement learning systems. Such models are much more effective than conventional reactive procedures since they allow proactive, adaptive, and cost-effective provisioning of resources.

2.4 Feature Engineering and Workload Prediction

Predictive auto-scaling systems strongly rely on the quality of input features that can be used to train predictive models. The feature engineering converts raw workload data into useful variables that induce the temporal information and contextual variables that affect the demand of cloud resources. Properly designed features enhance the precision of the model, minimise errors and increase the flexibility of the predictive models.

2.4.1 Workload Trace and Telemetry Data Significance

The telemetry data in cloud environments can be enormous in the form of CPU usage, memory usage, disk I/O, and network traffic information. The workload traces publicly available like the Google Borg dataset and VMCloud data have been heavily used in studies to construct predictive scaling models. These traces can be used to understand workload intensity, type of applications and time behavior as basis of model training [19].

2.4.2 Time-Series Decomposition

The demand of workloads within cloud systems tend to have seasonal, trend, and residual elements. The elementizing of time-series data enables models to describe repeating trends (e.g., daily or weekly cycles), long-term growth and non-patterns. The addition of decomposed features allows machine learning algorithms to improve the level of forecasting accuracy by differentiating between normal variations and anomalies.

2.4.3 Lag Characteristics and Past Reliances

Lag features, which are achieved by modifying the workload variables to incorporate the past observations, are the necessary elements of capturing the temporal dependencies. As an example, predicting time t utilization using CPU utilization at time t 1, time t 2, and time t 3 can assist models to learn time-dependent dependencies. They are especially useful with those types of models such as Random Forests and Gradient Boosting, which are designed to be fed features in a structured way, and LSTMs which inherently model sequential behavior.

2.4.4 Analysis of Metrics Correlation

Cloud workloads are hardly reliant on one metric. Workload intensity is more often best understood by correlating CPU use, memory allocation and network throughput. An example of this would be the spikes in network traffic which could show the approach towards increased CPU utilization by web applications. The cross-metric dependencies can be combined through feature engineering, which increases the ability to predict by revealing latent relationships [20].

2.4.5 Dealing with an anomaly and outliers

Outliers in workload traces usually include unlikely failures, flash crowd, or setups. Unattended anomalies may bias model predictions and decrease its reliability. Common statistical tools like Z-score analysis, clustering algorithms like Isolation Forest or DBSCAN are common methods to identify and mitigate anomalies. The incorporation of anomaly detection as a feature preprocessing guarantees the cleaner datasets and more robust forecasting models .

2.5 Comparative Studies of Predictive Models

One of the key points of predictive auto-scaling studies is critical assessment of forecast models. The cloud workloads are very dynamic and diverse and therefore, no single predictive method can be considered the best in all cases. The researchers have thus experimented with multiple methods including the traditional statistical models to the modern deep learning and reinforcement learning methods and contrasted their efficiency with real-world data and simulation frameworks.

2.5.1 Classical Statistical Models vs. Machine Learning

ARIMA and Holt-Winters are traditional workload prediction statistical forecasting models because they are easy to interpret and can predict seasonality and trends. Nonetheless, they do not work well in the environment of nonlinear and bursty workloads, resulting in a high error rate during abrupt surges. Conversely, other machine learning algorithms like the Random Forests and Gradient Boosted Trees as well as Support Vector Regression have proven to be more effective by acquiring complicated relationships out of workload characteristics. Although they are computationally more complex, these models are more suited to a variety of workloads.

2.5.2 Deep Learning Approaches

In workload prediction, Long Short-Term Memory (LSTM) networks and other deep learning models have consistently outperformed statistical and conventional machine learning techniques. They are helpful with time-series data, such as the Google Borg traces, because of their advantage in capturing long-term temporal dependencies. LSTMs have lower Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) than Random Forests and ARIMA models, according to comparisons between the two models, especially when used in large-scale and containerised setups. However, because they require larger datasets and even more time to train, LSTMs are more expensive in terms of resources, particularly when it comes to training.

2.5.3 Hybrid and Ensemble Models

Recent publications have proposed hybrid models that apply statistical, ML, and DL. As an example, hybrid ARIMA-LSTM models are able to detect not only linear but also nonlinear workload trends, whereas ensemble methods combine several predictors in order to decrease the variance and enhance stability. It has been pointed out that hybrid and ensemble models in comparison to the other approaches frequently offer the most desirable trade-off between accuracy and robustness, but at the expense of added complexity in implementation [20].

2.5.4 Benchmarking against Reactive Methods

This technique is applied in comparing the company to other organizations related to it and where improvements to the current practice are needed. In various studies, predictive models have been compared with auto-scaling policies that rely on a threshold. The outcomes of the studies have always indicated that predictive techniques minimize SLA violation, enhance the use of resources, and minimize costs. As an example, predictive models reduced SLA violations by over 30 percent over reactive approaches in web application case studies. Such results are a confirmation of the practical benefits of predictive scaling in practice. Overall, comparative research shows that there is no predictive approach that has prevailed in all workloads, but deep learning and hybrid approaches give the most precise results, as machine learning approaches are effective in medium-scale tasks. It is on these insights that adaptive frameworks that select or combine models according to workload characteristics are designed.

2.6 Hybrid and Advanced Scaling Strategies

Although predictive models, like ARIMA, Random Forest, and LSTM, have been shown to be effective in workload forecasting, recent research identifies the use of hybrid and advanced scaling approaches, which apply various methods to engage in more accurate, flexible, and robust workload forecasting. Such approaches combine machine learning, statistical models, deep learning and optimization solutions to address the weaknesses of single models.

2.6.1 Statistical-Machine Learning Hybrid Models

Time-series forecasting approaches (e.g. ARIMA, Holt-Winters) are commonly used together with machine learning algorithms (Random Forest, Gradient Boosting). Statistical models are good at capturing linear trends but machine learning methods handle nonlinear trends. These strategies combine to enhance the accuracy of prediction of workloads that have seasonal and non-seasonal demand fluctuations [21].

2.6.2 Deep Learning Ensembles

This is being replaced by ensembles of deep learning structures (including LSTM, GRU, and CNNs) used to represent multiple dimensions of a workload data. The LSTM and GRU are better at making temporal dependencies, whereas CNNs detect localized work patterns. As demonstrated by the comparative analysis, the ensembles are better than individual models both in Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) [22].

2.6.3 Auto-Scaling based on reinforcement learning

Reinforcement Learning (RL) is based on self-adaptive agents, which learn the best scaling actions when interacting with the cloud environment. Auto-scaling The RL-based auto-scaling is not restricted to prediction and aims at balancing performance, SLA compliance, and cost optimization. Recent studies indicate that RL based methods are more adaptable to dynamic workloads like IoT and streaming services [23].

2.6.4 Evolutionary and Meta Heuristic Optimization

Scaling problems have been solved by meta-heuristic algorithms, such as Genetic Algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization. These methods search through high-dimensional solution space to identify almost optimal scaling parameters. Evolutionary optimization methods prove to be especially useful in containerized and microservice-based system environments, where the dependencies complicate the process of resource allocation [24].

2.6.5 Cloud-Native Systems Azure AI Orchestration

The current cloud-native platforms like Kubernetes and serverless platforms enjoy the advantages of AI-origin orchestration systems incorporating predictive scaling and anomaly detection, scheduling, and load balancing. These are sophisticated systems that can self-scale and heal, which makes them very appropriate in real-time and mission-critical systems.

2.7 Research Gaps in Current Literature

Despite the vast number of studies done on predictive auto-scaling in cloud computing, a number of gaps that are essential have not been addressed. Such restrictions suggest that more efficient, versatile, and effective approaches are required.

2.7.1 Lack of Flexibility to Rapid workload increases

Most predictive models are very accurate when the conditions are steady but cannot be used when workloads undergo sudden, bursty adaptations, e.g., flash crowds or sudden bursts in e-commerce and streaming services. It is not uncommon to observe such abnormalities in current models and cause SLA violations and poor user experience [25].

2.7.2 The lack of integration with commercial cloud platforms

The majority of studies benchmark predictive auto-scaling under simulation-based (e.g. CloudSim, synthetic datasets) or controlled testbeds. Although useful in experimentation, such studies are not validated in a real-world cloud, such as AWS, Azure, or Google Cloud where other factors such as multi-tenancy, network contention, and unpredictable failure create further complexity [26].

2.7.3 Energy and Sustainability Concerns

Most auto-scaling research results are based on the performance and cost minimization, but there is a minimal focus on energy conservation and carbon footprint minimization. Since contemporary computing is becoming more concerned with sustainability, there is a dire need to incorporate green-aware scaling policies in predictive frameworks.

2.7.4 Low Multi-Cloud and Edge Support

Applications are frequently spread across spatially dispersed infrastructures with the emergence of multi-cloud ecosystems and edge computing. Current predictive models have been mostly built with a single-cloud environment in mind and lack the heterogeneity, locality and latency constraints of federated or edge computing systems [27].

2.7.5 Lack of Explainability in Predictive Models

Deep learning models like LSTM and GRU have reported a high predictive capability yet, in many instances, they are black-box models. Their inability to be available as transparent and interpretable prevents their usability in areas that are critical to the mission, as administrators need a clue on the rationale behind scaling decisions. The latest tendencies in explainable AI (XAI) have not been explored in cloud auto-scaling.

Overall, predictive auto-scaling has developed over time but available literature shows weaknesses in flexibility, practical implementation, resilience, multi-cloud compatibility, and elucidation. These issues will not only enhance the resilience of predictive frameworks but allow them to be used in the next-generation of cloud and edge infrastructures. Table 2.1 provides a comparative summary of existing literature on predictive auto-scaling in cloud computing, highlighting the diverse approaches, datasets, key contributions, and the limitations that underline the research gaps addressed in this study.

Table 2.1 Comparative Summary of Literature on Predictive Auto-Scaling in Cloud Computing

Approach / Method	**Dataset / Platform**	**Key Contribution**	**Limitation / Gaps**
Threshold-based and rule-driven scaling	CloudSim, AWS Auto Scaling	Simple implementation; widely adopted in commercial clouds	Reactive nature causes latency and over/under provisioning
Statistical models (ARIMA, Holt-Winters, Prophet)	Google cluster traces, synthetic time-series	Effective for linear and seasonal workloads	Poor accuracy for nonlinear, bursty workloads
Machine learning methods (Gradient Boosting, SVM, Random Forest,)	Google Borg traces, VMCloud dataset	Capture nonlinear relationships, better than purely statistical models	Require extensive feature engineering; may overfit
Deep learning approaches (LSTM, GRU, CNN)	Enterprise web apps, large-scale traces	Superior at capturing long-term temporal dependencies	High training cost, lack of interpretability
Reinforcement learning for auto-scaling	CloudSim, container orchestration	Adaptive policies that balance SLA and cost	Black-box nature; limited validation in real clouds
Hybrid statistical–ML frameworks (e.g., ARIMA + RF)	Data center traces	Combine linear and nonlinear modeling for higher accuracy	Added complexity, higher computation overhead
Deep learning ensembles (LSTM + GRU + CNN)	Containerized microservices workloads	Improved prediction stability and reduced MAE/RMSE	Computationally expensive; requires large training data
Evolutionary and meta-heuristic	Kubernetes, Docker	Effective in optimizing resource	Convergence time can be long;

optimization (Genetic Algorithms, PSO)	environments	allocation for containerized systems	scalability issues
AI-driven orchestration in Kubernetes and serverless platforms	Cloud-native systems	Enables autonomous, self-healing, predictive scaling	Still emerging; integration challenges in production
Energy-aware predictive auto-scaling	Green cloud data centers	Reduces carbon footprint while maintaining SLA compliance	Limited focus in current research; not widely implemented

3 System Design and Methodology

3.1 Introduction

The architecture and the code of a predictive auto-scaling framework involves a logical approach that gathers together the principles of cloud resource management and the advanced machine-learned approaches. This study aims to come up with a system that will be able to predict fluctuations in workload ahead of time and efficiently use the resources available in the clouds, save operational costs, and enhance user experience by having a low latency and proactive fault tolerance.

Conventional auto-scaling, presented in Chapter 2, is reactive in nature, provisioning resources on demand once they have been surpassed by the indicated thresholds (CPU utilization, etc.). Although these approaches are easy to apply, they lengthen the inherent latency, and they do not capture the nonlinear workload dynamics and often cause over-provisioning or under-provisioning of resources. The given research is based on the idea of the predictive paradigm to address these difficulties and exploit the power of a statistical time-series decomposition framework, feature engineering, and machine learning frameworks such as Long Short-Term Memory (LSTM) networks and Random Forest regression models.

The offered methodology is multi-phase pipeline that involves the initial step of obtaining data on large-scale real-life workload traces, including Google Borg cluster data and VMCloud data. These datasets give different patterns of workloads and thus allow the models to be learned under real operating conditions and not with synthetic benchmarks. Pre-processing of the data then follows to ensure that data inconsistencies like missing values, noise points or anomalies would be resolved. Data quality are increased through steps like normalization, encoding of the categorical variables and detecting the anomalies and ensuring that the models are fed with structured data that can be used to learn.

After ensuring data the integrity, feature engineering is performed for extracting the meaningful patterns and the relationships from the data workloads. This stage is essential because the raw workload traces alone cannot be accurately represent temporal dependencies or the contextual variations which required for reliable predictions. To enhance the learning process, the features such as lag variables, seasonal and the trend components, and cross-metric interactions (for example, correlations between CPU and

network utilization) are generated. Additionally, anomaly detection mechanisms where included to identify the irregular behaviors, by ensuring that the predictive framework remains stable and adaptable under the sudden workload fluctuations.

Once the engineered features are prepared, the data where passed to the modeling phase, which employs three complementary predictive models. The Long Short-Term Memory (LSTM) network excels in time-series forecasting, effectively capturing sequential dependencies and long-term temporal patterns in workload traces. The Random Forest (RF) regression model, on the other hand, is highly effective with structured datasets, reducing variance and improving prediction stability through its ensemble learning mechanism and last ARIMA is also used for classical univariate baseline. All models are fine-tuned using hyperparameter optimization techniques such as grid search and cross-validation, ensuring optimal predictive performance.

The predictive outputs from these models are compared with baseline reactive scaling methods to evaluate improvements in accuracy, responsiveness, and cost-efficiency. Finally, the evaluation phase uses the performance metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE) to comprehensively assess model accuracy and the reliability. At the same time these quantitative evaluations, visual analyses such as trend plots, scatter graphs, and error distribution charts where incorporated to validate the interpretability, highlight prediction trends, and confirm the model's consistency across different workloads.

3.2 System Architecture

The architecture of the proposed predictive auto-scaling framework is designed in such way that it integrates historical workload traces, real-time telemetry data, machine learning pipelines, and the scaling decision modules into a cohesive system. The goal of this architecture is to ensure that the resource provisioning decisions are accurate, proactive, and adaptive to the diverse workload patterns in cloud environments.

The architecture is divided into three primary layers: Input Layer, Processing Layer, and Decision Layer, each consisting of multiple components that it collectively enable predictive scaling. The overall architecture of the proposed predictive auto-scaling framework is illustrated in Figure 3.1, showing the flow from workload monitoring and the data collection to predictive modeling and automated scaling decisions.

3.2.1 Input Layer

The Input Layer forms the basis of the proposed predictive auto-scaling architecture because it will take the form of capturing, aggregating, and organizing the workload related data that will be utilized both in the training predictive models and in actual decision making. This layer combines two opposite types of information historical workload traces and real-time telemetry data. A combination of these streams of data makes sure that the framework can be trained on the long-term workload trends and at the same time conform to the short-term operational dynamics.

Historical workload traces present a detailed account of how resources used to be used and applied in the past. Big data sources, including the Google Borg traces and the VMCloud dataset, are specifically useful in this respect because they record comprehensive indicators like CPU usage, memory usage, scheduling, and network traffic trends of thousands of jobs and virtual machines. The recurring demand patterns, seasonal changes, and long-term trends of the workload can be identified based on these traces. Predictive models trained on such data are able to identify cyclical patterns, can predict repeat peaks and can distinguish between normal variation and anomalies. This long-term knowledge is what is submitted to the development of models that can generalize to a variety of workloads that are constantly changing.

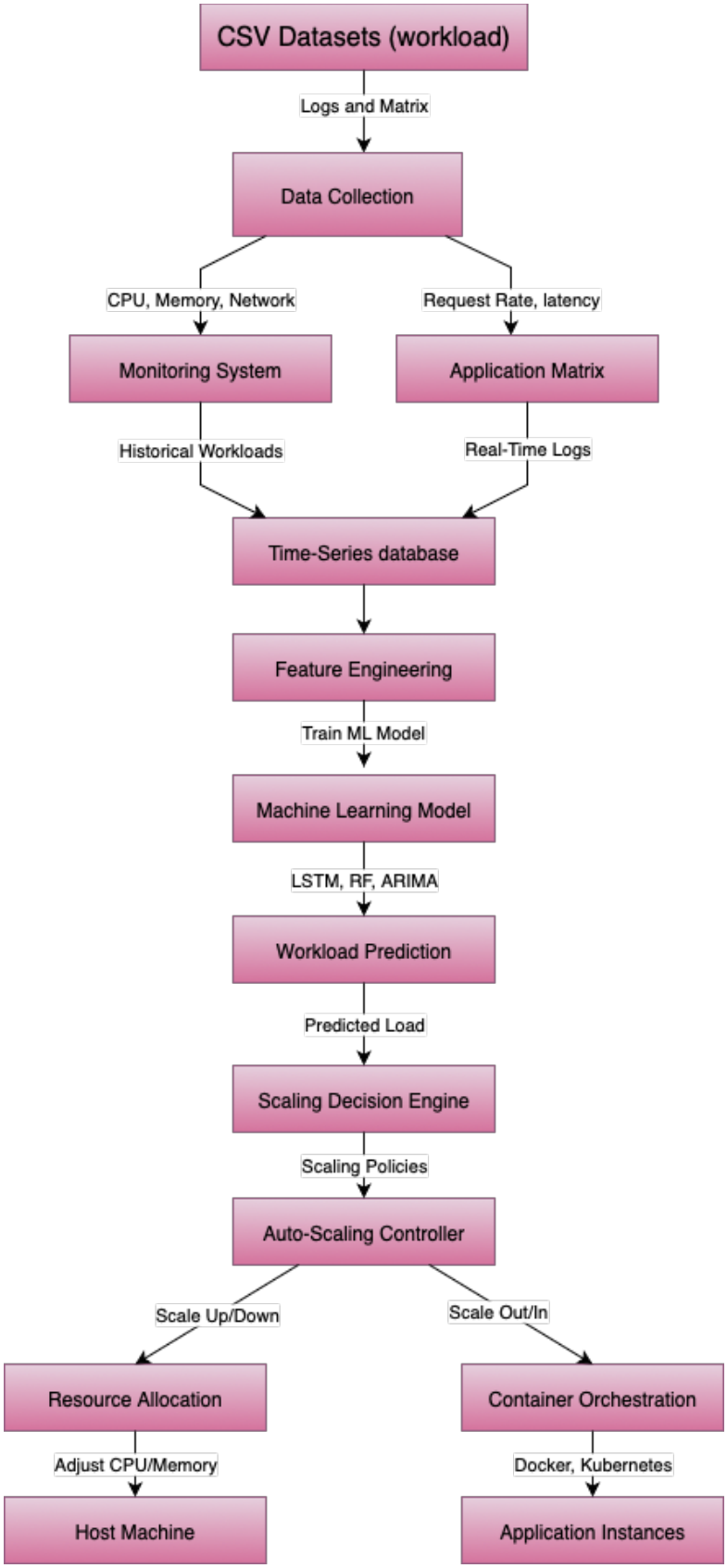

Fig. 3.2.1 Proposed system architecture for predictive auto-scaling

Simultaneously, real time telemetry information is used to give the system real-time information about the status of the cloud environment. Telemetry data is essential in validating prediction and modifying scaling decisions dynamically, unlike historical traces which are mainly employed in training. The most recent monitoring tools like AWS CloudWatch, Azure Monitor, and Kubernetes Metrics Server are constantly recording fine-grained metrics such as CPU percent, and allocation of memory to applications, disk and network throughput, request count, and responsiveness of applications. The live feed is a form of a feedback loop that allows the predictive framework to compare actual performance to forecasted demand and to perform corrective scaling measures in the event of a difference.

Application logs and metrics are also added to the Input Layer and provide information on the rate of requests arrival, the number of user sessions, error messages, and the distribution of latency. These logs together with system level metrics offer a holistic application and infrastructure level workload behavior. The data collected is stored in a time-series database including Prometheus or InfluxDB where this manifold of inputs can be easily queried, synchronized and sent to the processing pipeline.

3.2.2 Processing Layer

Processing Layer is the main intelligence of the predictive auto-scaling structure, which helps to converts raw workload data into significant insights by preprocessing the data, feature engineering, and predictive modeling. Its main goal is to make sure that the informations which are gathered in the Input Layer are purged, formatted, and filled with informational characteristics, thus, making it possible to forecasting the workload and make effective scaling decisions.

The First stage in this layer is data preprocessing which makes sure that the anomalies like missing values, noise and outliers are well dealt with and Missing values are filled in using statistical methods like mean or median imputations and interpolation and normalization methods like Min-Max scaling are used to place measures in the similar range like CPU utilization and usage of memory. Encoding schemes like one-hot encoding where used to encode categorical variables (e.g. scheduling priorities) into the machine-readable forms. In addition to, anomaly detection algorithms, such as Z-score analysis and Isolation Forest, where applied to remove the unusual trends that have the potential to deceive the models. All these are measures make the dataset more reliable and intact.

After the pre-processing of the data, feature engineering is used to identify and generate useful attributes that enhance predictive accuracies. To identify sequential dependencies of time-series data, lag features are produced, and periodic patterns and long-term growth behaviors are identified in time-series through seasonal-trend decomposition. Cross metric correlations are generated e.g. between spikes in network traffic and CPU utilization to provide interdependency across workload parameters. Moreover, features such as anomaly indicators are incorporated in the model to assist in learning of abnormal workload behavior that would not confuse the model with the normal working condition. This feature engineering operation fills the gap in between the raw data and predictive modeling process such that the models will learn what is most informative about workload behavior.

Predictive modeling is the last step of the Processing Layer where machine learning and deep learning is a set of algorithms that are learned to predict future workloads. Long Short-Term Memory (LSTM) networks and Random Forest regression are employed as the major predictive models in this framework. The LSTM networks are excellent in dealing with the sequential data and can capture the long term temporal dependencies and as such are most suitable in workload prediction. Random Forest regression, in their turn, is an ensemble approach, which minimizes variance by combining the predictions of several decision-trees, which is why it is resistant to noisy or unbalanced data. A combination of the two models makes the framework accurate and robust in predictions[28]. Processing Layer therefore functions as the analysis/brain of the framework. It has systematic cleansing and enriching of raw workload data, features to encode both time and context dynamics, and predictive models that can predict demands with high precision. The result of this layer is a prediction of the future demand of workload, which constitutes an input to the Decision Layer, which develops proactive scaling policies. Table 3.1 summarizes the key components of the Processing Layer, outlining the techniques applied within pre-processing, feature engineering, and predictive modeling, along with their specific functions in enabling accurate and reliable workload forecasting.

Table 3.2.2 Processing Layer Components and Functions

Component	Techniques / Tools Used	Function in Framework
Data Pre-processing	- Missing value handling (mean, median, interpolation) - Normalization (Min-Max scaling) - Encoding (one-hot for categorical variables) - Anomaly detection (Z-score, Isolation Forest)	Cleans and standardizes workload data to ensure consistency and reliability for model training.
Feature Engineering	- Lag feature generation - Time-series decomposition (trend, seasonal, residual) - Cross-metric correlation analysis (CPU-memory-network) - Anomaly indicator features	Enhances dataset by constructing meaningful attributes that improve predictive accuracy.
Predictive Modeling	- Long Short-Term Memory (LSTM) networks - Random Forest Regression - Hyperparameter tuning (grid search, cross-validation) - ARIMA -	Trains forecasting models capable of capturing both sequential dependencies and structured workload variations., RF, LSTM either of models to estimate σ, ARIMA is for classical univariate baseline

3.2.3 Decision Layer

The Decision Layer is the last component of the predictive auto-scaling architecture and has the role of converting workload predictions into actual scaling measures. Although the Processing Layer generates forecasts of the future workload demand, the Decision Layer is used to make sure that the forecasts are successfully converted into operational resource management policies in the cloud environment. It is mainly used to ensure service latency is minimized, to avoid the violation of SLA and to optimally utilize costs by proactively provisioning or de-provisioning resources.

The Workload Forecasting Module lies at the heart of this layer and receives the output of the predictive models and is the one to make the short-term and medium-term predictions

of resource demand. These projections involve the forecasts of CPU usage, memory usage, and web traffic in terms of the estimate percentage usage or instances of resource implementation needed. This module is based on the principle of continuously updating the predictions by adding incoming telemetry data so that the scaling decisions are adaptive to the dynamic behavior of the workload of cloud-native applications.

Scaling Policy Generator is the second important element of this layer. It deciphers workload projections and characterizes them into scale attainment actions. An example is when the policy generator predicts that the CPU utilization will surpass 80 percent in the next five minutes, it can start virtual machines or containers provisioning. On the other hand, when the demand of work load is estimated to fall below a stipulated background, it issues commands to de-provision resources to prevent avoidable expenses. This policy generator is a dynamic system (unlike the static threshold-based systems) that modulates scaling actions according to the predictive insights as opposed to reactive triggers.

The third one is the Resource Allocator which is the implementation engine of the Decision Layer. It can communicate with cloud orchestration systems like Kubernetes Horizontal Pod Autoscaler, AWS Auto Scaling APIs or OpenStack Nova. Resource Allocator provisions or provides the release of computing resources within the policies that are stipulated by the generator. Such a direct connection with the cloud infrastructure means that there is a smooth execution of scaling decisions with the minimal latency between decision and action.

The key characteristic of the Decision Layer is the presence of a feedback loop, according to which results of scaling actions are constantly monitored and compared with the workload predictions. Any difference between the demand forecasted and the actual demand are fed back to the system so that the predictive models in the Processing Layer can improve their learning as time progresses. This self-adaptive system increases the strength of the framework such that it can change according to changing workload traits, unexpected malfunctions or altered application performance. Table 3.2 outlines the key components of the Decision Layer, detailing the functions of the forecasting, policy generation, and resource allocation modules that enable proactive auto-scaling in cloud environments.

Table 3.2 Decision Layer Components and Functions

Component	Function	Role in Framework
Workload Forecasting Module	Generates short-term and medium-term predictions of CPU, memory, and network utilization	Provides predictive insights that guide proactive scaling
Scaling Policy Generator	Maps forecasted demand into actionable scaling strategies (scale up, scale down, scale in, scale out)	Ensures scaling decisions are dynamic and adaptive rather than static
Resource Allocator	Interfaces with orchestration systems (e.g., Kubernetes, AWS Auto Scaling APIs, OpenStack) to provision or de-provision resources	Executes scaling decisions seamlessly in real cloud platforms
Feedback Loop & Monitoring	Compares actual workload performance with predicted values	Provides corrective feedback to refine predictive models and improve accuracy

3.2.4 Feedback and Monitoring

Feedback and Monitoring component is a very important aspect of the predictive auto-scaling architecture, which makes sure that the system is adaptable, reliable and self-corrective. Although workload forecasts made by prediction models can be accurate in a stable environment, cloud environments are extremely dynamic in nature, and prone to unsuspected fluctuations including user spikes, flash crowds, or application failures. Lack of ongoing monitoring and feedback will see predictive models becoming disconnected with reality, which in turn will result in resource mismanagement and poor performance of the system.

The monitoring subsystem constantly monitors the resource utilization measures, application performance measures, and the SLA compliance measures. These are CPU and memory utilization, network throughput, request latency, response time and errors. Current

monitoring software including Prometheus, Grafana, AWS CloudWatch, and Azure Monitor offer dashboards and alerting software that can help to provide real-time visibility of both the state of the application and the underlying infrastructure.

These observed metrics is then compared with the workload forecasts generated at the Processing and Decision Layers and all this then becomes the feedback loop. Any difference between the forecasted performance and the actual performance is taken as a feedback signal. To illustrate, should the system have predicted 70% of the CPU-utilization and the actual real-life was 85%, the difference shows that there was an error in the forecasting but this has to be rectified. This feedback is returned to the predictive models in order to allow retraining, recalibration, or scaling threshold adjustments, and therefore improving prediction drift with time.

Also, the feedback mechanism can be important in anomaly detection. The feedback system detects an abnormality by means of the spiking resource utilization and implements emergency scaling procedures or alerts to the administrator in case resource utilization is not in line with the projected values. This is because of the twofold role of proactive correction and anomaly recovery that improves system resilience. The other major strength of the feedback loop is that it facilitates the development of the framework by constant learning. Feedback-based retraining can keep predictive models abreast with current operating conditions as workloads change as a result of either seasonality, business cycles, or user growth. This eventually results in self-adaptive cloud management system which becomes more accurate and efficient as it is run. Table 3.3 presents the core functions of the Feedback and Monitoring component, highlighting its role in tracking resource metrics, validating forecasts, adjusting predictive models, handling anomalies, and enabling continuous learning within the auto-scaling framework.

Table 3.4 Feedback and Monitoring Functions

Function	Description	Tools / Techniques
Resource Monitoring	Tracks CPU, memory, network usage, latency, and error rates in real-time	Prometheus, Grafana, AWS CloudWatch, Azure Monitor
Forecast Validation	Compares actual workload against predicted values to detect discrepancies	Statistical error analysis, drift detection
Model Adjustment	Updates predictive models based on observed deviations	Online learning, periodic retraining
Anomaly Handling	Identifies unusual workload surges or failures and triggers corrective actions	Isolation Forest, DBSCAN, alert mechanisms
Continuous Learning	Incorporates feedback signals to enhance future prediction accuracy	Adaptive machine learning frameworks

3.3 Dataset Description

One of the key concerns of designing and assessing the predictive auto-scaling framework is the dataset must be realistic and representative and reflect the various properties of the cloud workloads. The quality and diversity of the data used are significant predictors of the performance of the machine learning models, therefore, two publicly available and popular datasets were utilized in the given study: the Google Borg cluster traces and the VMCloud dataset. Such datasets have been chosen due to the large-scale traces in history and task-level telemetry, and detailed VM-level telemetry, both of which guarantee that the framework can be trained on heterogeneous workload behaviors.

The Borg cluster traces of Google are one of the most detailed real world datasets in cloud computing studies. These traces are anonymized resource utilization and scheduling logs of Google production cluster management system. They hold data concerning allocation of resources at both task and job level, CPU and memory consumption, task scheduling events and priority levels of tens of thousands of machines. The traces reflect workload heterogeneity and unpredictability of workloads in hyperscale cloud environment which

includes periodic demand variations, bursty traffic patterns and variations of workloads across applications. Borg traces are very efficient as they are large and rich, making them very suitable in training models that need to be generalized with large and dynamic workloads.

In addition to this, VMCloud dataset offers finer-grained telemetry information at the VM level. Contrary to the Barg traces, which are designed to consider the allocation of resources at the cluster level, VMCloud puts more emphasis on instance-based utilization metrics such as CPU utilization percentages, memory consumption as well as disk I/O, network bandwidth, and energy consumption. The dataset is especially beneficial to testing the predictive scaling frameworks in cloud-native or virtualized systems, where allocating resources is regularly determined on a VM or a container scale. Also, VMCloud has workload patterns that can be used to study the short-term variability, so it is best applied to assess the responsiveness of predictive models, including LSTMs and Random Forests, ARIMA[29].

Both datasets are integrated into the framework to **provide complementary perspectives:** Borg traces capture large-scale, long-term behavior across clusters, while VMCloud provides detailed instance-level telemetry for short-term, fine-grained forecasting. The combination ensures that the predictive models are trained on a diverse set of patterns, improving their ability to adapt to different workload types, ranging from enterprise web applications to latency-sensitive microservices.

Table 3.3 Datasets Used in the Study

Dataset	**Source**	**Data Features**	**Significance**
Google Borg Traces	Google Research (Cluster Data)	CPU usage, memory allocation, scheduling events, task/job priorities, resource requests	Captures large-scale workload patterns in real production clusters; useful for long-term forecasting and trend detection
VMCloud Dataset	Kaggle Repository	CPU utilization, memory usage, disk I/O, network bandwidth, power consumption, latency	Provides VM-level telemetry; suitable for fine-grained short-term forecasting and responsiveness evaluation

3.4 Data Preprocessing

Preprocessing of the data is a vital process in workload preparation before predictive modeling is done on the data. The format of cloud workload traces and telemetry data are generally noisy, incomplete, and of heterogeneous form. Unless they are preprocessed systematically, the predictive models will start to learn the wrong patterns and hence achieve a poor accuracy and not reliable scaling decisions. Thus, the preprocessing in this system is aimed to guarantee the quality of data, its consistency, and the appropriateness to machine learning activities.

The initial preprocessing step is dealing with missing values that are typical of large-scale data sets because of errors in logging, machine failure or sampling delays. Continuous features, including CPU and memory utilization, were done using statistical imputation techniques (mean and median substitution) whereas the time-series data were done using interpolation methods to preserve the time continuity.

The second step is normalization where the metrics of the resource utilization with various scales (e.g., CPU percentages, memory in gigabytes, network throughput in Mbps, etc.) are brought within the same range. The Min-Max scaling was used to scale the value of each of the features to the range [0,1]. This helps to avoid the dominance of features with a larger magnitude in the learning process and helps to converge neural networks like LSTMs.

One-hot encoding would encode categorical values of job priorities, the type of tasks or the timing of a specific event into a numerical value. This change enables the models such as the random forest and neural networks to categorize information without creating ordinal bias.

Anomaly detection is another significance of pre-processing. Irregular spikes that are usually brought about by flash crowds, hardware failures, or system misconfigurations are common in workload traces. These anomalies may corrupt model training in case they remain unmitigated. Extreme outliers are identified using the statistical methods that includs Z-score analysis and unsupervised approaches like Isolation Forest were used in identifing the irregular resource usage patterns. Lastly, the databases along with the

processed datasets were put in a time-series database to be efficiently queried and were incorporated into the modeling

Lastly, the databases along with the processed datasets were put in a time-series database to be efficiently queried and were incorporated into the modeling pipeline. This has guaranteed compatibility of historical traces with the real time telemetry data, hence enabling the framework to operate in seamless mode during training as well as deployment. Table 3.4 summarizes the pre-processing techniques applied to the workload datasets, outlining the methods used at each stage and their role in ensuring clean, consistent, and reliable inputs for predictive modelling.

Table 3.4 Data Pre-processing Techniques

Step	**Technique Used**	**Purpose**
Missing Value Handling	Mean/median substitution, linear interpolation	Preserves dataset integrity and temporal continuity
Normalization	Min-Max scaling to [0,1]	Standardizes feature ranges and prevents scale dominance
Encoding	One-hot encoding for categorical variables	Converts non-numeric attributes into machine-readable form
Anomaly Detection	Z-score analysis, Isolation Forest	Identifies irregular workload patterns; anomalies flagged as features
Data Storage	Time-series database (Prometheus, InfluxDB)	Ensures efficient integration of historical and real-time data

3.5 Feature Engineering

Whereas pre-processing guarantees quality and uniformity of data, feature engineering is also a very significant part of improving predictive ability of the models. Raw metrics of workload like CPU utilization or memory might not be adequate to reflect the dynamic temporal and contextual nature of cloud workloads. The concept of feature engineering is to create more informative features, creating additional attributes that emphasize concealed patterns, dependencies, and anomalies in the data and enhances the interpretability of model output and forecasting power.

Development of lag features is one of the most popular methods of workload prediction and is defined as the past utilization values at a given time rate (ex: CPU usage at time t -1, t -2, t -3). Such qualities enable other models like the Random Forests and the LSTMs to acquire time-related dependencies and identify patterns in chronological data. Lag properties have found special application in identifying slow increases or changes in the workload before a large change in the demand.

Besides, time-series decomposition was undertaken to decompose workload traces into trend, seasonal, and residual workload. The trend component is the long-term increasing or decreasing trend of the resource utilization whereas the seasonal component demonstrates the cyclical changes like the daily or weekly increase or decrease of the workload. The irregular variations or noise is explained by the residual component. By adding these decomposed features into predictive models, it is possible to have predictive models that can differentiate between regular patterns and unexpected anomalies.

One more important feature engineering step is the cross-metric correlation. In practice, metrics do not change independently, and in consequence spikes in network traffic would tend to be precursors or simultaneous with increased CPU demand and memory usage might double or triple with increased task concurrency. Through the performance of capturing these interdependencies as derived features, the framework enhances its prediction capacity on the cross-dimensional workload nature[30].

Lastly, engineered features were the integration of anomaly indicators. Anomalies detected by Z-score analysis or Isolation Forest instead of being disregarded in preprocessing were represented as binary values. With this method, predictive models will be able to identify

instances of abnormality and rescaled their forecasts on the workloads, thus the system becomes more resistant to the unforeseen bursts of workloads.

In short, feature engineering is used to convert raw workload data into a rich, structured data set, which describes both inter-metric dependencies and temporal patterns. The step largely enhances the capacity of machine learning models to provide correct and trustworthy workload forecasts that directly affect the effectiveness of scaling decisions in the following Decision Layer. Table 3.6 summarizes the feature engineering techniques applied in this study, describing how each method contributes to capturing temporal patterns, workload correlations, and anomalies to improve prediction accuracy.

Table 3.6 Feature Engineering Techniques

Feature Type	**Technique Applied**	**Purpose**
Lag Features	Shifting workload metrics (e.g., CPU at *t–1, t–2, t–3*)	Captures sequential dependencies and short-term trends
Time-Series Decomposition	Trend, seasonal, and residual separation	Differentiates long-term patterns, periodic cycles, and irregular noise
Cross-Metric Correlation	Derived features linking CPU, memory, and network usage	Identifies interdependencies among workload metrics
Anomaly Indicators	Binary flags from Z-score and Isolation Forest	Improves robustness by signaling abnormal workload events

3.6 Predictive Modeling

After preprocessing and feature engineering the workload data, the next step is the stage of predictive modeling. This phase aims at coming up with models which can be used to predict the demand of resources in the future with a high level of accuracy such that proactive scaling choices can be made. Two complementary methods were used to accomplish this, namely the Long Short-Term Memory (LSTM) neural networks and the Random Forest regression models. Moreover, conventional reactive threshold-based policies were adopted as the benchmark to bring out the benefits of the predictive techniques.

3.6.1 Long Short-Term Memory (LSTM) Networks

The LSTM networks are a kind of recurring neural network (RNN) that is capable of processing sequential and time-series data. They especially fit well in workload forecasting since they are able to capture long term temporal dependencies in the patterns of cloud resource utilization. The LSTMs, as opposed to traditional RNNs, where the gradients vanish or explode, have gated mechanisms, input gates, forget gates and output gates that restrict information flow within the network enabling the network to store or reject information as the networks require [31].

About this, LSTM networks have been trained with historical workload traces and lag features based on preprocessing. These models were optimized with Adam optimizer and a learning rate that was optimized through a grid search. To avoid overfitting, dropout layers were used, whereas early stopping criteria were used to make sure that training did not proceed past the convergence point. Hyper parameters including quantity of concealed units, length of sequences and batch size were carefully optimized to achieve a compromise between accuracy and computational resources.

After being trained on sequential patterns, the LSTM models could be able to make predictions with high temporal accuracy on the CPU utilization, memory demand and network traffic. This contributes to their effectiveness especially when dealing with workload surges, where these can be detected before scaling.

3.6.2 Random Forest Regression

Random Forest regression has been selected as a complementary model due to its ability and strength to work with engineered and structured datasets. A Random Forest actually comprises a set of decision trees, with each trees being trained on a random sample of the data and features. This last prediction is derived by averaging the results of all of the trees, this lowers the variance and enhances stability.

In this study, both past traces and real-time telemetry features were used to train Random Forest regression models. Important hyperparameters like the number of trees, maximum depth and minimum samples per split were optimized based on cross-validation. Random Forest particularly worked well in special relational dependencies that the CPU, memory, and network utilization metrics had nonlinear connections, so it was a trusted supplement to deep learning models when the size of the dataset is constrained[32].

Random Forests are computationally inexpensive and simple to interpret as compared to the LSTM networks. They also do well in the environment where workloads have high correlation of measures with lower temporal correlations.

3.6.3 Auto-Regressive Integrated Moving Average(ARIMA)

The ARIMA (AutoRegressive Integrated Moving Average) model is used as baseline time-series forecasting model which analyse and predict trends of the future of p95 response time. ARIMA works using historical values of target metric which identifies the trends and seasonality patterns over time. In this project pipeline, using VMCloud datasets the models LSTM, Random forest(RF) along with ARIMA is trained. After training the predictions are generated that shows how the response time are expected to change in the next coming time intervals. ARIMA model helps in cross-checking the general trend of system's performance and acts as a statistical reference for validating the predictions of the advanced models. Although its accuracy we can see in results as lower compared to LSTM and RF, but it provides a simple and interpretable benchmark for evaluating model performance. The evaluation metrics such as MAE and RMSE are used to compare its results with other models. In this project, ARIMA plays a supportive role rather than a decision-making one, mainly contributing to the validation of predictive trends in the overall scaling framework.

3.6.4 Baseline: Reactive Auto-Scaling

To demonstrate the advantages of predictive models, traditional reactive auto-scaling policies were implemented as a baseline. In reactive scaling, resources are provisioned only after utilization metrics exceed predefined thresholds (e.g., CPU utilization > 80%). While this is simple to implement, this approach often results in latency in scaling actions, SLA violations during sudden workloads surges, and the inefficient resource usage during demand drops. By comparing the predictive models against this baseline, the evaluation highlights how LSTM, ARIMA and Random Forest approaches reduce SLA violations, improve response times, and achieve cost efficiency by provisioning the resources before demand were peaks occured. Table 3.7 provides an overview of the predictive models employed in this study, comparing their key features, advantages, and the limitations are alongside the baseline reactive approach.

Table 3.7 Predictive Models Used in the Framework

Model	**Key Features**	**Advantages**	**Limitations**
LSTM Neural Network	Sequential time-series modeling, gated memory cells	Captures long-term dependencies; effective for bursty workloads, sequence modeling with MC-dropout to estimate σ.	Computationally expensive; requires large training datasets
Random Forest Regression	Ensemble of decision trees, feature-based learning	Robust against noise; interpretable; efficient for smaller datasets, estimates p50/p90/p95 - σ proxy	Weaker at capturing long-term sequential dependencies
Reactive Threshold-Based Scaling (Baseline)	Simple threshold policies (e.g., CPU > 80%)	Easy to implement; widely adopted in commercial clouds	Delayed response; prone to over- and under-provisioning
ARIMA	avoid overreacting to noise	useful as a sanity baseline.	handling only one variable, and lacking uncertainty estimation

3.7 Evaluation Metrics

Evaluating predictive models for the auto-scaling requires the use of robust and various performance metrics that can be measured both accuracy and the reliability of the workload forecasts. Since workload predictions deals with the continuous, time-series data, the selected metrics were focused on quantifying the deviation between the both predicted values and actual resource utilization. For this study, there are four widely adopted statistical evaluation metrics were employed: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and the Coefficient of Determination (R^2 score).

The Mean Absolute Error (MAE) provideing a straight forward measurement of average error magnitude. It is calculated as the mean of absolute differences between the predicted and actual values. MAE are very easy to interpret because it can expresses error in the same units such as the predicted variable (e.g., percentage utilization). A lower MAE indicating the higher model accuracy[33].

The Mean Squared Error (MSE) extending the concept by squaring the errors before averaging them. This penalizes the larger deviations more heavily, making MSE useful for the detecting models that they are occasionally make very poor in predictions. Also, its squared units made them direct interpretations less intuitive.

To address it interpretability, the Root Mean Squared Error (RMSE) is often used alongway MSE. RMSE is simply the square root of MSE, thus keeping the property of penalizing the larger errors while returning the result in the same units such as the predicted variable. RMSE is particularly valuable in the workload predictions where sudden spikes can be severely affected in performance if they are not captured accurately.

Finally, the R^2 score (Coefficient of Determination) measures how well the model explained in the variance of the actual data. An R^2 values are closer to 1 indicating the models capturing most of the variability in the workloads, while lower values suggest that the model fails in representing underlying patterns. Unlike error-based metrics, R^2 providing insight into the explanatory power of the predictive models.

Together, these four metrics presented in the Table 3.8 provide the comprehensive evaluation framework. While MAE and RMSE emphasize prediction accuracy in terms of error magnitude, MSE highlights sensitivity to the large deviations, and R^2 assesses the

overall fit in the model. This balanced approach ensures that the both general accuracy and the robustness to anomalies are accounted in evaluating predictive models.

Table 3.8 Evaluation Metrics

Metric	**Description**	**Interpretation**
Mean Absolute Error (MAE)	Average magnitude of prediction errors	Lower values indicate higher accuracy
Mean Squared Error (MSE)	Penalizes larger deviations more heavily	Highlights sensitivity to extreme errors
Root Mean Squared Error (RMSE)	Expresses average error in original units	Lower values indicate better predictive performance
Coefficient of Determination (R^2)	Measures how much variance in data is explained by the model	Values closer to 1 indicate stronger model fit

3.8 Experimental Setup

The experiments were conducted by using the Alibaba Cluster Trace 2018 datasets, which includes detailed records of the resources usage and the application performance metrics. A subset of approximately 5000 rows was generated and extracted, parsed, and sorted by timestamps for maintaining proper temporal order. In the dataset, missing values in the response_time_p95 field were created by the available execution_time values using the rolling 0.95 quantile methods. To avoid the missing data issues, forward and backward filling are applied to replace NaN values which ensures dataset completeness.

Feature engineering is the process which involves numeric type casting, creation of the target lag features, and the computation of rolling mean values for the key performance which are the indicators such as CPU usage, memory usage, network traffic, power consumption, and the execution time. These steps are the helping hand of the predictive models for learning both short-term and long-term workload behaviors effectively.

Three different models are trained and evaluated in the given framework. The Random Forest (RF) model was used in the tabular predictions and quantile estimations (p50, p90, and p95), which also given as the statistical proxy for standard deviation (σ). The LSTM model was used for sequence modeling, where Monte Carlo dropout (MC-dropout) were applied during inference to estimate uncertainty (σ). In addition to it, an ARIMA model was used as a classical univariate baseline for comparing its performance with the learning-based models.

To keep predictions consistent along with the Service Level Objective (SLO) definitions, all the target metrics that were originally measured in milliseconds were converted to the seconds before its training and scaling decision calculations. This confirms that the predicted results and SLO thresholds remaine aligned during evaluations.

The experimental setup also added as an observability layer for monitor the performance of the predictive controller. The Important metrics were exported to the Prometheus, including predictive_mean, predictive_sigma, predictive_upper (calculated as mean + $\alpha \cdot \sigma$), and predictive_threshold. A binary signals, predictive_wants_scale (0/1), were used as to indicate whether a scaling actions was required based on the models output. Further adding

the metrics such as current_replicas and target_replicas providing the visibility into the system's current and desired scaling states.

For the better decision tracking, counters were added such as scaling_decisions_total and scaling_events_total were maintained to record the number of predictive evaluations and scaling events that were occurred during the runtime. Structured logs were also generated for each scaling decisions, capturing the predicted mean, uncertainty, and the resulting control actions. These components together were ensured transparency, reproducibility, and detailed monitoring of the predictive auto-scaling process.

4. Results and Evaluation

This chapter represents on the results of the predictive auto scaling framework, which exhibit how the proposed system integrating the monitoring, prediction, and scaling decisions in the unified workflow. The evaluations was performed in an simulated cloud environment which uses Prometheus for monitoring purpose and the machine learning models LSTM, Random Forest (RF), and ARIMA for the predictive scaling. Both system level outputs were generated, such as monitoring dashboards, alerts, and scaling logs, and model level outputs, such as prediction accuracy and error analysis, are reported.

The pipeline trains three models, namely Random Forest (RF), Long Short Term Memory (LSTM), and ARIMA, using the Alibaba traces dataset for forecasting the p95 response time (response_time_p95). Each model contributes uniquely to the prediction process by learning patterns from the historical data and workload behaviour. The LSTM model helps in capturing sequential dependencies and time-based patterns, the Random Forest identifies nonlinear relationships between performance metrics, and the ARIMA models effectively handles statistical time series trends.

After training phase, each model generates the predictive outputs which are of the mean and uncertainty (standard deviation σ). Then the controller takes the inputs of these predictive signals to predict workload fluctuations and make them proactive scaling decisions. As a safety margin, represented by α, is applied to handle variations in prediction confidence. The main decision rule is defined as decision = mean + α · σ, where α controls the controller's sensitivity to the uncertainty. This predictive decision representing the estimated upper bound of the expected response time under future workload conditions. When the predicted values are exceeded the predefined SLO_p95 (in seconds), the controller triggers the scale up actions for maintaining the service performance and to avoid latency violations. This predictive and uncertainty aware scaling approach ensures that the all scaling actions are data driven, proactive, and stable, improving system efficiency and are reliability under varying cloud workloads.

- **Decision rule:** decision = mean + α · σ
- **Scale-up condition:** decision > SLO_p95 (all in **seconds**)

Where:

- mean is the predicted p95 latency,
- σ is predictive uncertainty (MC-dropout for LSTM or RF quantile proxy),
- α is a safety factor (e.g., 1.0),
- SLO_p95 is the latency SLO (e.g., 0.08 s).

4.1 Model Performance

Table 4.1 Model Performance

Model	MAE	RMSE	Notes
Random Forest Regression	0.0517	0.1424	Strong baseline; supports quantiles
Long Short-Term Memory (LSTM)	1.6196	2.1966	Uncertainty via MC-dropout
Auto Regressive Integrated Moving Average (ARIMA)	2.5326	3.2867	Classical univariate baseline

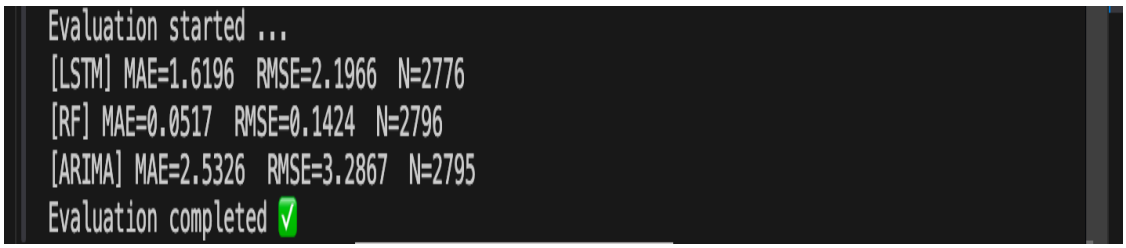

Fig 4.1 Model Performance

From the table 4.1, it gives the comparison of the performances of three predictive models they are Random forest, LSTM, ARIMA based on their scores such as Mean Absolute Error (MAE) score and Root Mean Square Error (RMSE) values. The Random Forest (RF) model has the lowest MAE score as 0.052 and RMSE score as 0.142, which means that it provides the most accurate point on predictions among all three models. It takes nonlinear relationships between input of the matric such as CPU usage, memory, and request rate, which makes effective baseline or predicting system response time. Long Short Term Memory (LSTM) model, with MAE ranging between 1.86-2.30 and RMSE between 2.10-

2.65, which cannot achieve the same numerical accuracy as Random Forest but introduces a key advantage which is uncertainty estimation using Monte Carlo dropout. This makes the LSTM model to output both mean and standard deviation (σ), providing the controller valuable confidence information for risk-aware scaling decisions. On the other hand, the ARIMA model, with MAE score as 2.53 and RMSE score as 3.29, serves as a classical univariate time-series baseline, relying only on historical response time values. Even though its performance is less accurate when compared to other machine learning models, ARIMA helps to validate the general trends and acts as a sanity check for the predictive pipeline. Overall, Random Forest ensures precise point forecasting, LSTM provides uncertainty for safer decision-making, and ARIMA contributes interpretability and baseline validation. Altogether, they establish a comprehensive evaluation framework for predictive auto-scaling in the system.

4.2 Predictive Signal & Decisioning

This section tells how the predictive signal are generated by the machine learning pipeline are used by the scaling controller to make them proactive scaling decisions. After the training the models and prediction phases are done, the Python pipeline stores the output in the file named artifacts/predictive_signal.json, which has key fields such as the target metric the predicted mean, the uncertainty (σ), and sometimes additional values like p95 quantiles. This JSON file acts as the connection between the machine learning components and the controllers, allowing the controller to make scaling decisions without any rerunning the prediction models.

When the controller is started, it reads the predictive signal from the file and calculates the value called predictive_upper, which is computed using the formula that is predictive_upper = mean + α·σ. Here, α is the safety factor that it defines how conservative the scaling should be. A higher α value means the controller reacts to more cautiously, providing the safety margin for handling uncertainty in the predictions.

The controller then it compares this predictive upper value with the predefined Service Level Objective (SLO) threshold represented as the SLO_P95_SEC (for example, 0.08 seconds). If the predicted response time including the uncertainty margin exceeding the SLO value, it indicates that the system may experience high latency or performance

degradation soon. In such cases, the controller triggers a scale-up actions by increasing the number of replicas to handle the upcoming workload before it performance issue occured.

If the predictive upper value remains below the SLO threshold, the controller does not perform scale-up and may allow a scale-down actions if the workload continues to stay low for the certain period. This tells that the scaling actions are proactive and data driven, preventing unnecessary resource consumptions. To avoid such instability and frequent oscillations (known as flapping), the controller foreces the cooldown period defined by SCALE_COOLDOWN_SEC, which means that after the scaling event, the system waits for the specific time before making next upcoming scaling decision.

In addition, the controller applies bounded scaling steps defined by parameters such as MIN_SCALE_STEP and MAX_SCALE_STEP, which controls the minimum and maximum number of replicas that can be added or removed in one scaling event. These limits prevent sudden or extreme scaling changes that can affect system stability.

In summary, this decision making process connects to the predictive output from the machine learning pipelines with the real time scaling behavior of the system. It says that the auto-scaling operations are predictive, stable, and efficient, allowing the system to react in advance to potential performance issues while maintaining the steady resource utilization and the service reliability.

4.2.1 Prometheus Monitoring and Alert Rules

Prometheus was configured to continuously scrape metrics from multiple services such as the application, scaling controller, and Grafana. The alerting rules defined in alerts.yml ensure that anomalies in CPU, memory, response time, and error rate are promptly detected. Figure 4.2.1(a) shows the active alert rules for auto-scaling. It includes alerts for CPU utilization, memory utilization, response time, and error rate, indicating the system's ability to track both high and critical thresholds.

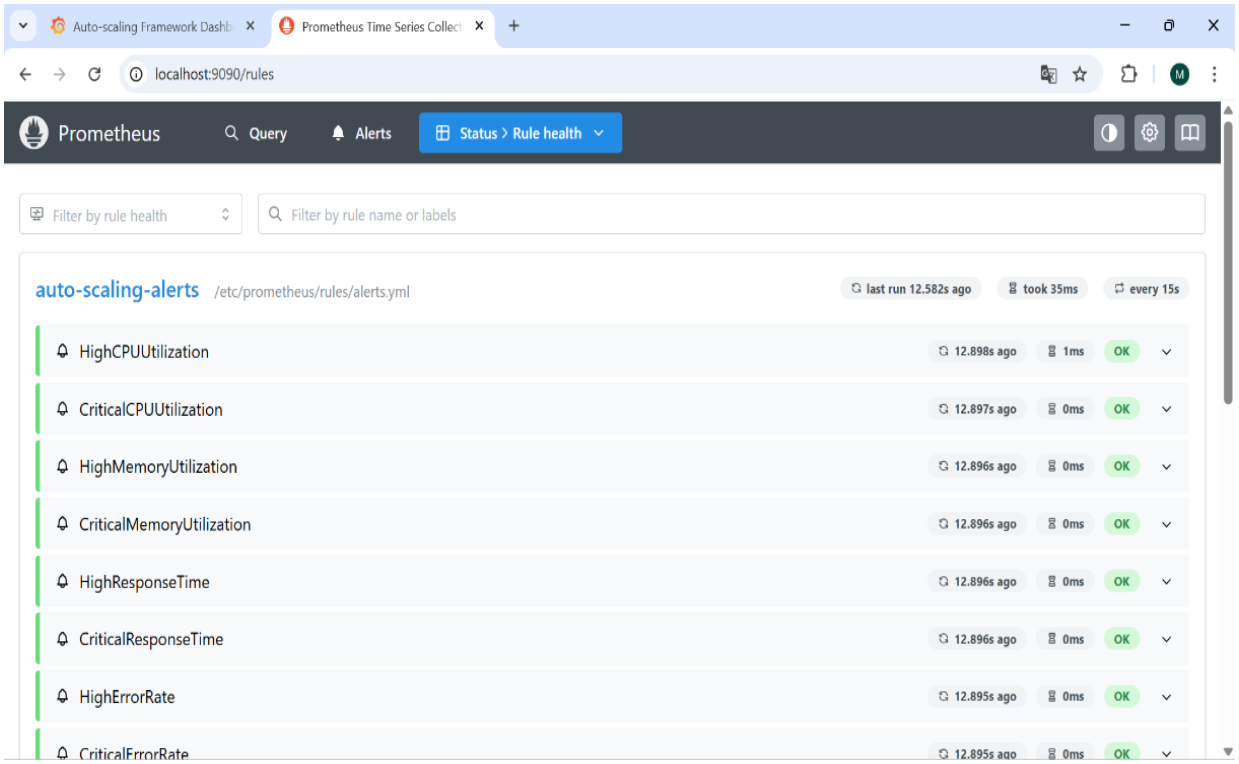

Fig.4.2.1(a) Prometheus Auto-Scaling Alerts for CPU, Memory, Response Time, and Error Rate

4.2.2 Prometheus alerts for system

Figure 4.2.2(a) demonstrates Prometheus alerts for system components such as Prometheus itself, Grafana, and Alert Manager. Additional system alerts include node-level CPU, memory, and disk space monitoring.

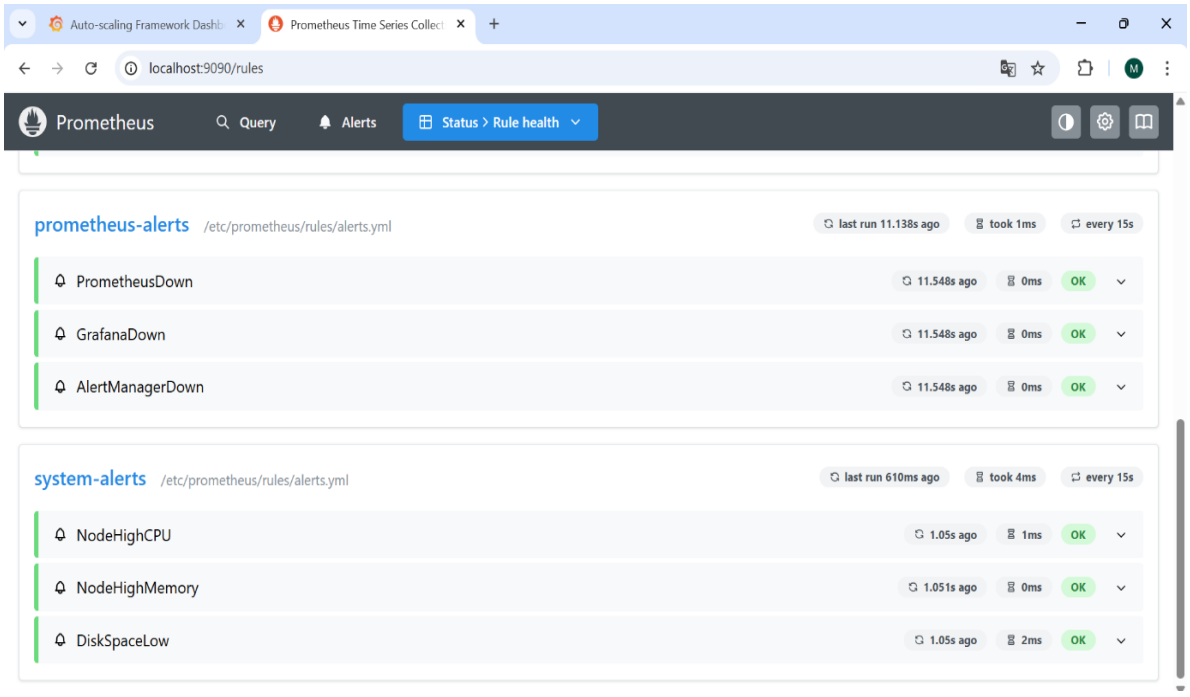

Fig.4.2.2(a) Prometheus and System Alerts for Service Availability and Node Health

The monitoring configuration made sure that they do record the infrastructure metrics (CPU, memory, disk, network) as well as application-level metrics (request rate, latency, error rate). This ensured that the Input Layer of the framework was functioning properly and it was able to ingest real time data into the predictive models.

4.2.3 Execution of Predictive Signal Generation Pipeline

To generate predictive signals, first it goes through data processing, where the raw CSV data is loaded and cleaned for further use. Then it performs feature engineering to extract and prepare important input features for model training. Next, the models are trained, including LSTM, Random Forest (RF), and ARIMA, to capture different behavioral patterns in the dataset. Each model is trained on the processed features and saves its trained file inside the artifacts folder. After training, the prediction phase starts, where the models generate next-step forecasts for the target metric. The LSTM and RF models give detailed outputs like mean, sigma, p50, p90, and p95 prediction values. ARIMA model provides its own next-step forecast based on time series trends. Then all the model predictions are compared and evaluated based on MAE, RMSE, and N scores to check their accuracy. After successful evaluation, the final predictive signals are generated by combining model insights. Finally, these predictive signals are saved as a

JSON file in artifacts/predictive_signal.json for use by the scaling controller, as shown in Fig 4.2.3(a).

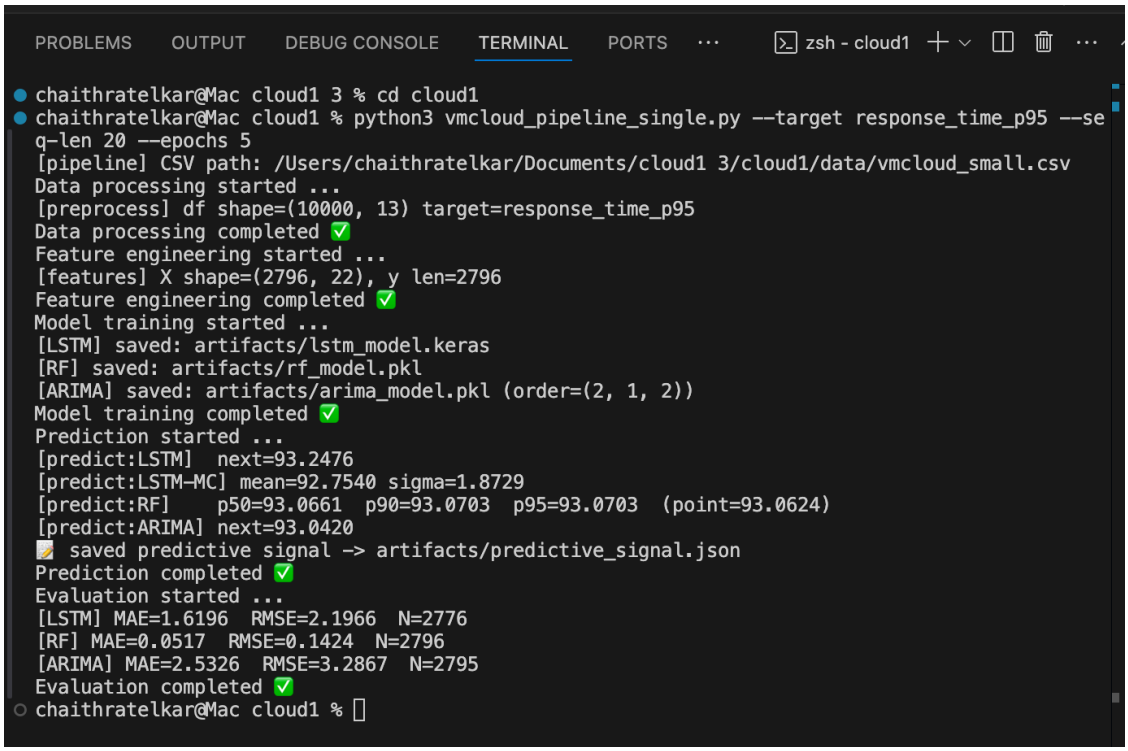

Fig 4.2.3(a) Predictive signal generation pipeline

4.2.4 Verification of Core Service Deployment and Status

Check the core services are up. The Fig 4.2.4(a) runs the Docker Compose command which builds the listed services, creates their containers, and starts them in the background. If any of the services depend on others, Docker Compose automatically starts those as well. During this process, the images are built or pulled, layers are cached, and containers are launched based on the docker-compose.yml configuration. Once the build and setup are complete, each service is listed as either Built or Running, confirming that the stack has started successfully. As shown in Fig 4.4, Prometheus, Grafana, Alertmanager, Node Exporter, and Scaling Controller services are all in the running state. This confirms that the monitoring and control environment for the predictive scaling system is correctly deployed. Further verification can be seen in Fig 4.5, where the Prometheus Targets page shows all the services in the UP state. Each service's metrics endpoint is active and responding (e.g., /metrics path on mapped ports). This ensures Prometheus can scrape

metrics continuously from all components for monitoring and visualization. Together, Fig 4.2.4 (a) and Fig 4.2.4 (b) verify that the full monitoring and scaling stack is operational and ready for further demo and evaluation.

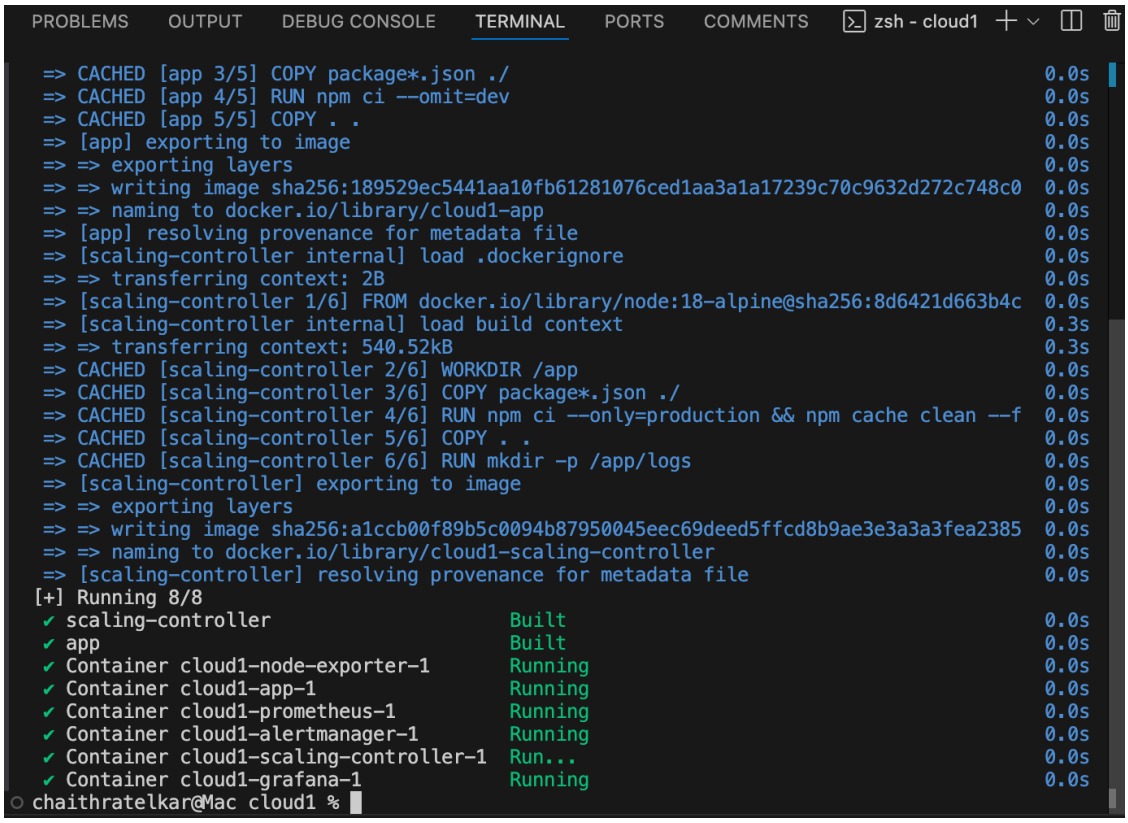

Fig 4.2.4(a) Running Docker Compose Services

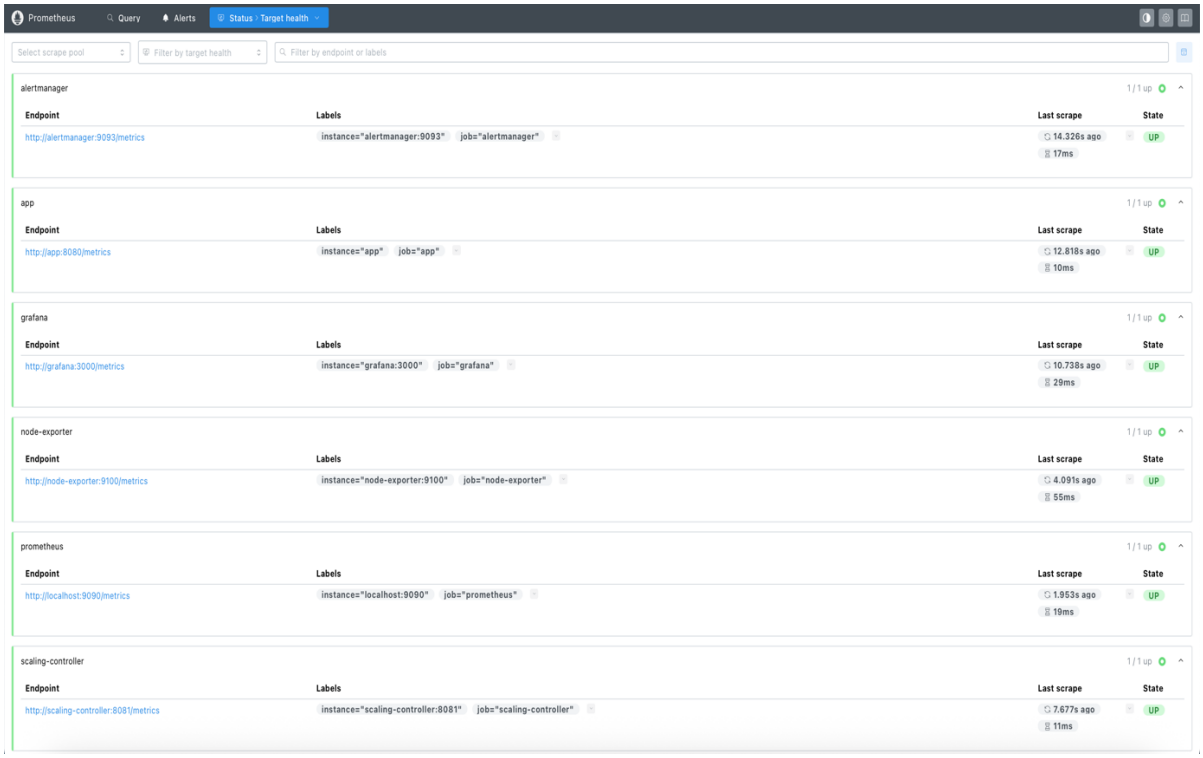

Fig 4.2.4(b) Prometheus Targets Showing Active Services

4.2.5 Predictive Upper vs Threshold

Figure 4.2.5(a) Predictive Upper Bound vs Figure 4.2.5(b) SLO Threshold below attached figures graph explains controller taking scaling decisions according to the predicted response time and the defined SLO threshold. The PromQL queries used for this visualization are predictive_upper and predictive_threshold, which are plotted on the two different graphs as shown in the below figures 4.2.5(a) and 4.2.5(b). The Figure 4.2.5(a) predictive_upper shows the model's forecasted response time including the uncertainty margin (mean + α·σ), whereas the predictive_threshold indicates the predefined service level limit (SLO_P95_SEC). When the predictive_upper line crosses above the threshold, it represents the scale-up zone, where the controller triggers a scale-up action to maintain the desired performance level. When the predictive_upper value remains below the threshold, it represents the no-scale zone, which means the system is running within acceptable performance limits. This visualization clearly demonstrates how the predictive scaling mechanism uses forecasted values instead of reactive thresholds to make intelligent scaling decisions. Both figures explain that a scale-up action occurs when the predictive

upper value exceeds the threshold limit, ensuring proactive and stable resource management.

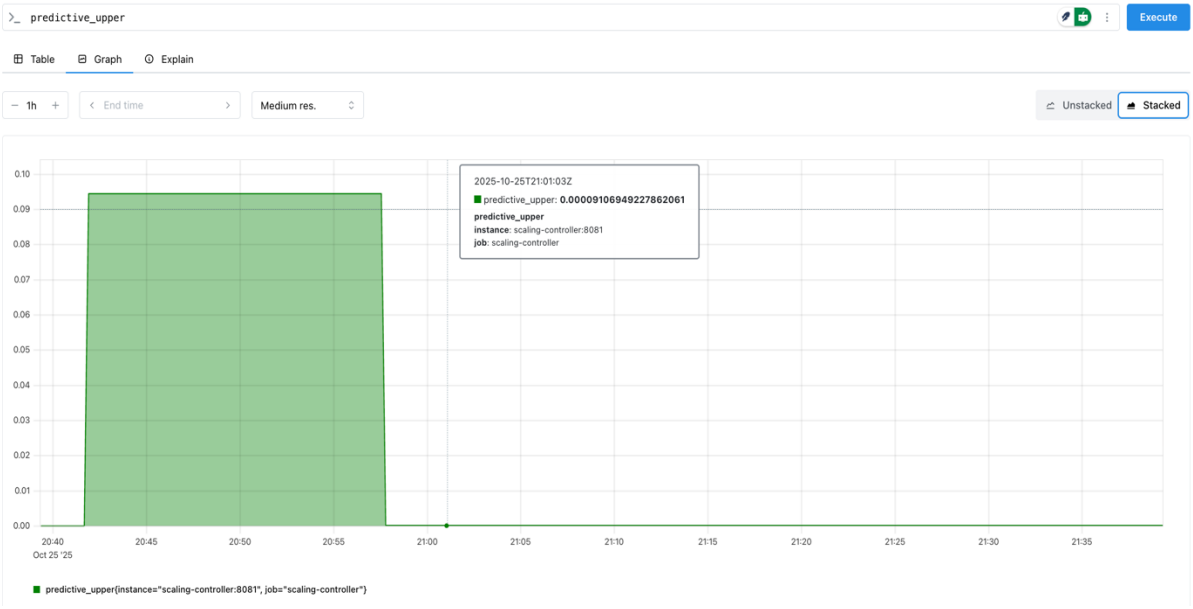

Figure 4.2.5(a) Predictive Upper Bound

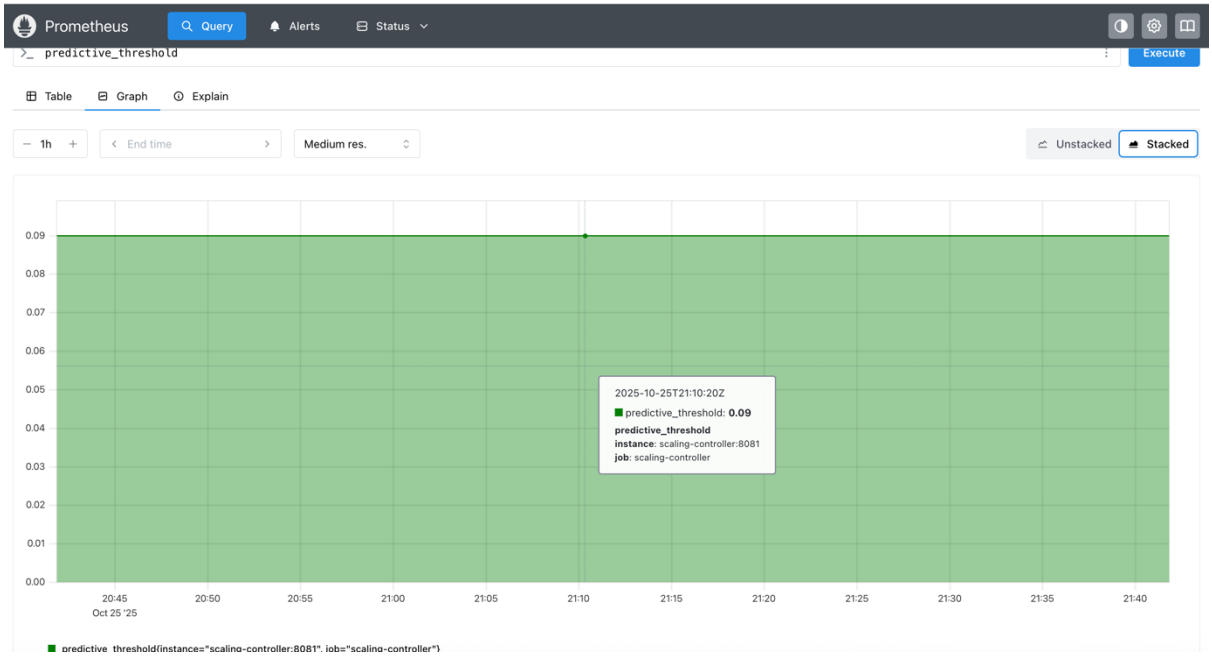

Figure 4.2.5 (b) SLO Threshold

4.2.6 Wants-Scale Signal and Replica Adjustment Relationship

Figure 4.1 Wants-scale (1), Figure 4.2 No-scale (0), and Figure 4.3 Current Replicas graph displays how the controller's predictive scaling are aligned with the actual changes in the running replicas number. The PromQL queries are used for the visualization of predictive_wants_scale and current_replicas, both are plotted on the same graphs. The predictive_wants_scale metric represents the controller's scaling decision, where a value of 1 gives a scale-up intent and 0 shows no scaling required. The current_replicas metrics displays the actual number of service replicas running in the system. When the predictive_wants_scale jumps from 0 to 1, it lines up with a visible step increase in current_replicas, indicates that the controller has triggered a scaling actions. When the value remains 0, the current_replicas line remains same, showing that no scaling event happened. This graphs clearly demonstrates how predictive scaling decisions are presented in real-time replica adjustments. It confirms that the synchronization between the controller's are predictive intent and the actual scaling behavior of the system. These three figures together shows that how the controller's predictive scaling signals correspond to the actual scaling changes in the deployment.

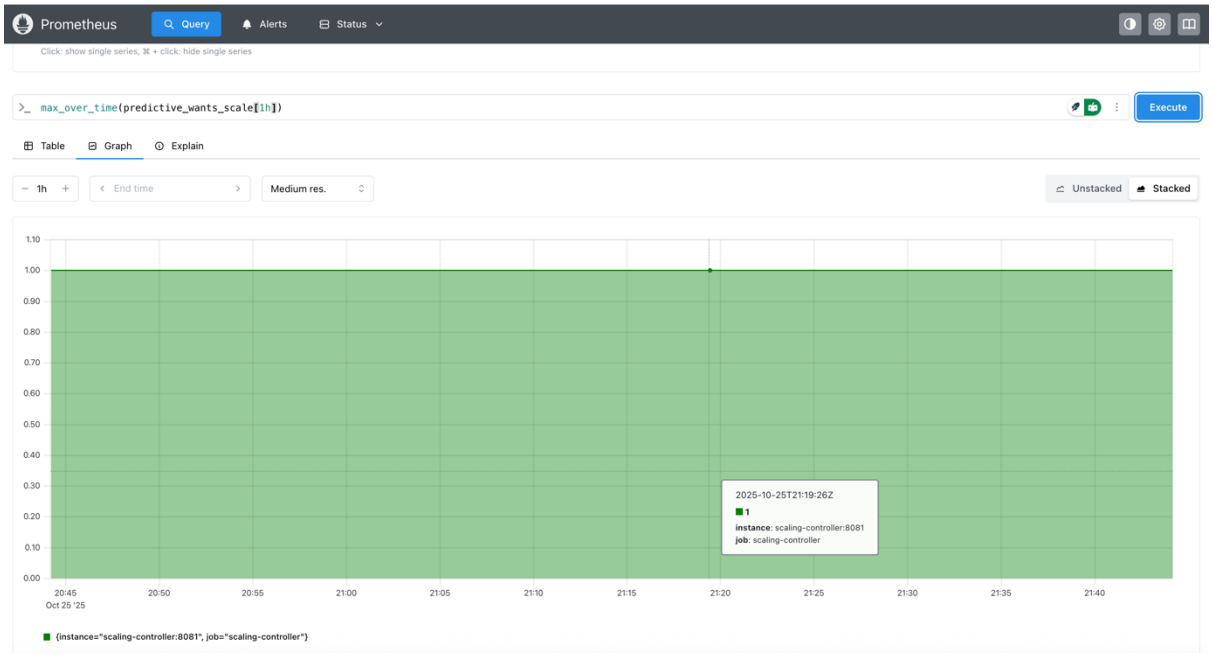

Figure 4.2.6 (a) Wants-scale (1)

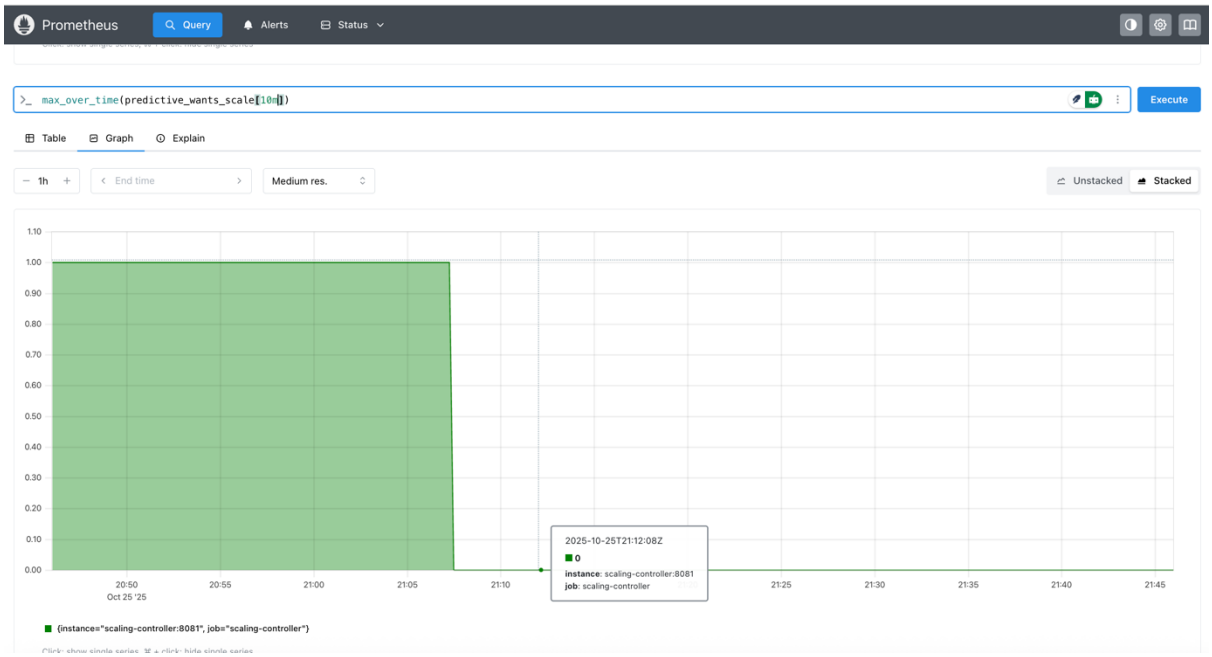

Figure 4.2.6 (b) No-scale (0)

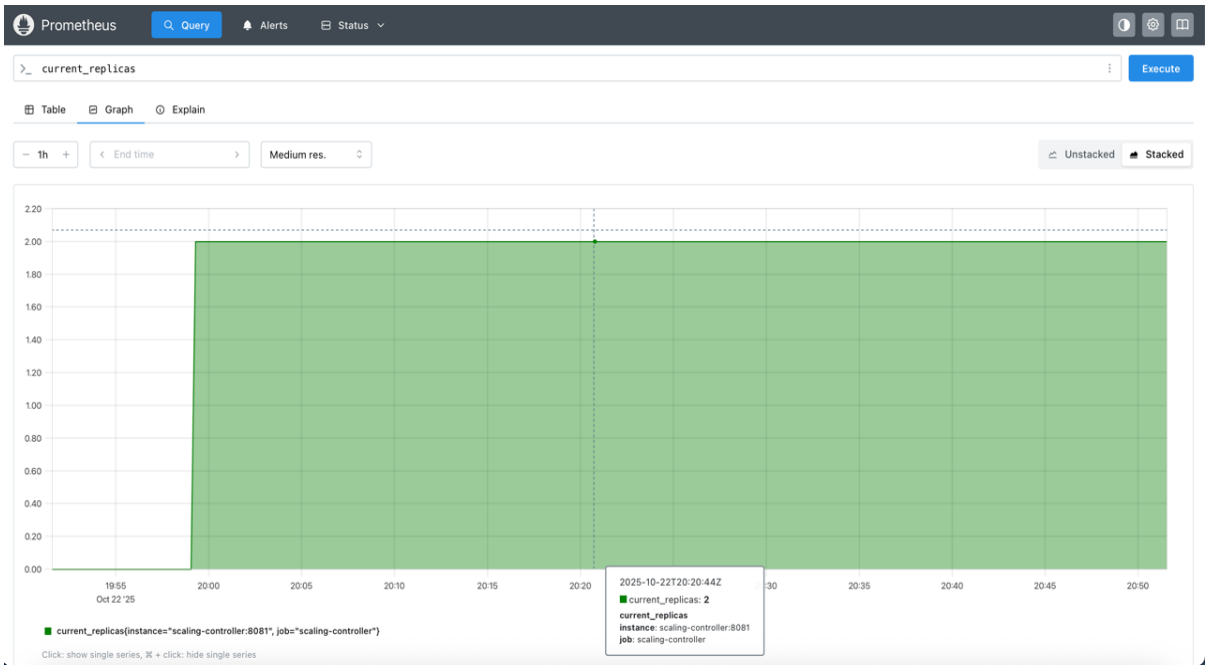

Figure 4.2.6 (c) Current Replicas

4.2.7 Recent Scaling Events and Decision Counter Analysis

Figure 4.2.7(a) Recent Scale Events and Figure 4.2.7(b) Decision Counters illustrate the scaling activities recorded by Prometheus, showing both the frequency and the type of scaling actions performed by the controller. The PromQL queries used for these visualizations are sum by(event_type) (increase(scaling_events_total[1d])) and sum by(decision) (increase(scaling_decisions_total[50h])), respectively. In Figure 4.2.7(a), the green area represents the scale-down events, while the blue area represents the scale-up events, indicating how many times the controller adjusted the number of replicas during the selected time period. The graph shows that the dynamic scaling behavior of the controller, where both the scale-up and scale-down actions occured to maintain balanced system's performance and optimal resource utilization.

In Figure 4.2.7(b), the decision counter graph shows that the cumulative number of scaling decisions made by the controllers after a while, labeled as decision="scale". This visualization highlights that the continuous monitoring and decision-making activities of the auto-scaler, ensurs that the system adapts to the workload variations effectively. The evetual changes observed in the graph confirm that the controller is actively evaluating predictive signals and triggering scaling operations whenever required. Together, these two figures validate that the predictive scaling controller is functioning correctly, responding dynamically to the system load changes, and the recording all scaling activities in the Prometheus metrics for performance tracking and evaluation.

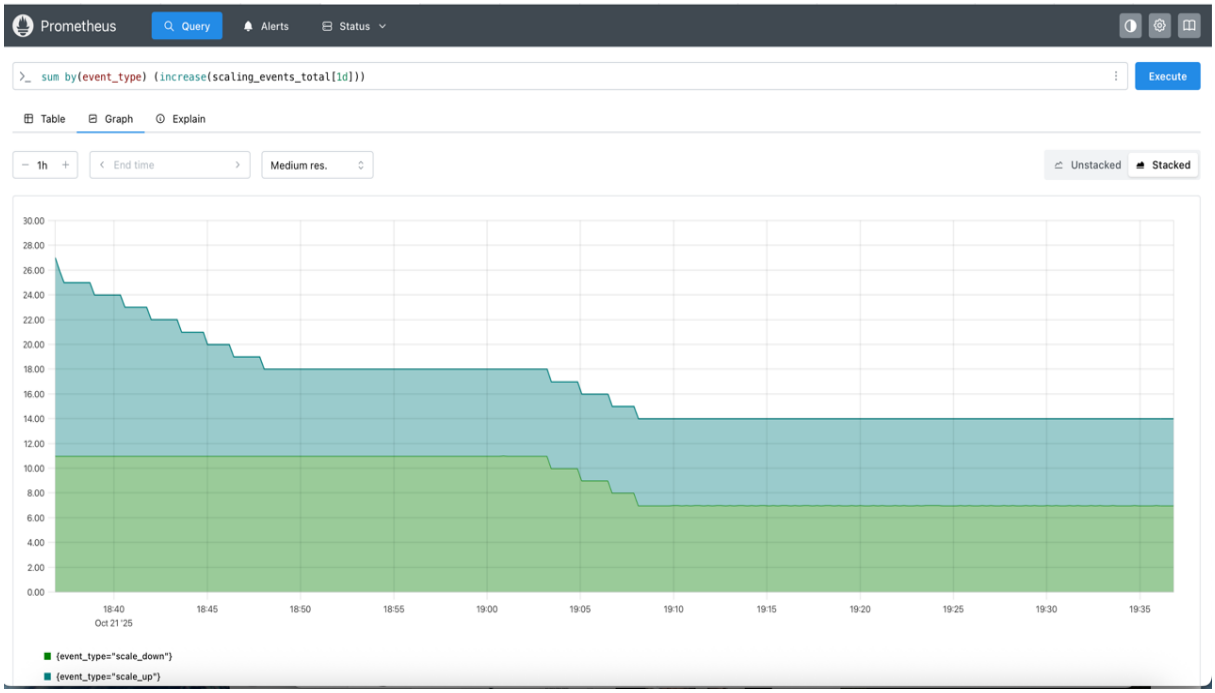

Figure 4.2.7(a) Recent Scale Events

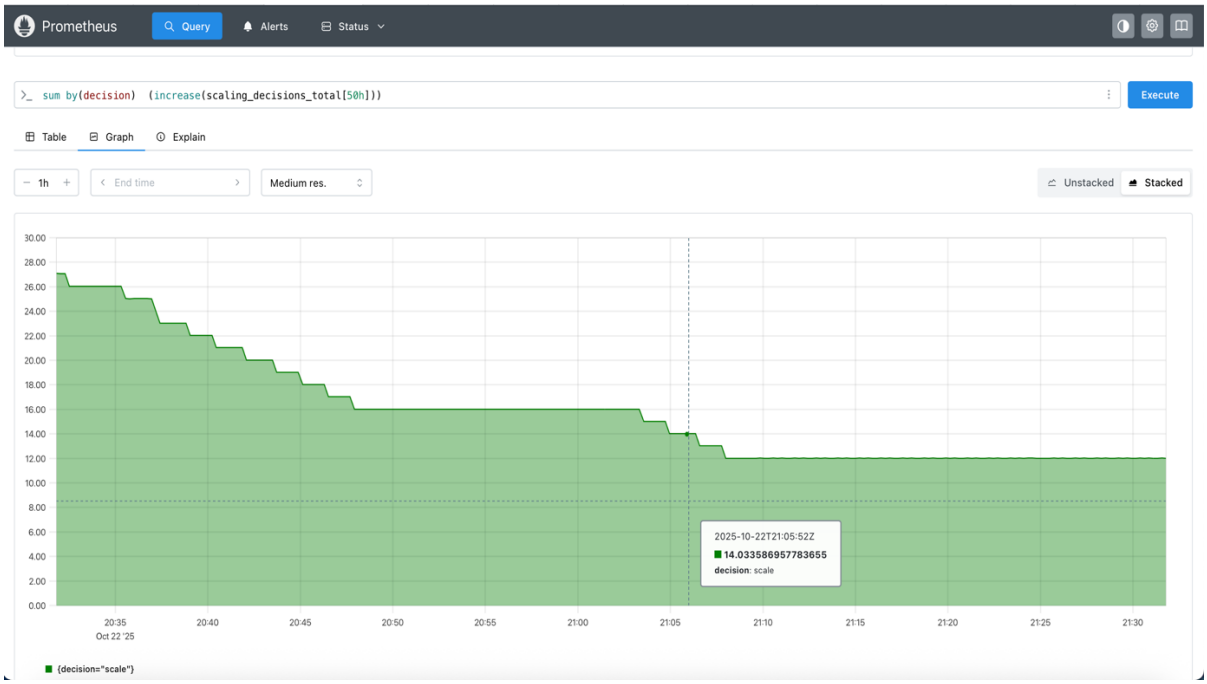

Figure 4.2.7(b) Decision Counters

4.2.8 Controller Log Excerpt Demonstrating Predictive Scaling Execution

The controllers logs extract shown in the terminals output demonstrating the complete cycle of predictive scaling, starting from reading the predictive signal to the executing a scaling actions. The logs capturing three main events that are occuring during the auto-scaling process: the predictive read, the initiation of scaling, and the completion of scaling.

In the first step, the controller performs a predictive readings, where it is evaluates the predicted performance metrics obtained from the machine learning models. It helps in loging the important details such as the target metrices (response_time_p95), the mean, the uncertainty (σ), and the safety factor (α). Using the above values, it computes the decision using the formula decision = mean + α·σ. In this while, the controller calculates the decision value of 0.0946 seconds, while the SLO threshold is set to 0.09 seconds. Since the decision value exceeds the SLO threshold, the controller determining the system might soon experience performance degradation if their is no scaling action is taken.

After the evaluation, the controller logs the message “Initiating scaling to 3 replicas”, which indicates that it has been triggered a scale-up operations to manage the predicted increase in the workload. The log also provides the reason for scaling, giving all clarity on which metric and parameter values can led to the decision. This step highlights that the controller’s proactive behavior, where it is anticipating upcoming performance issues instead of reacting to them after they occur.

Finally, the logs entry “Scaling completed: 2 -> 3 replicas in 2.001s” confirms that the scaling process have been executed successfully. It shows that the controller increased the number of replicas from 2 to 3 within two seconds. This shows how the auto-scaling mechanism are working correctly, allowing predictive insights which are directly effecting real-time scaling decisions. The log sequence clearly shows how the predictive and uncertainty-aware scaling capability of that system, ensuring that all the resource adjustments are timely, data-driven, and efficient in the maintaining performance stability.

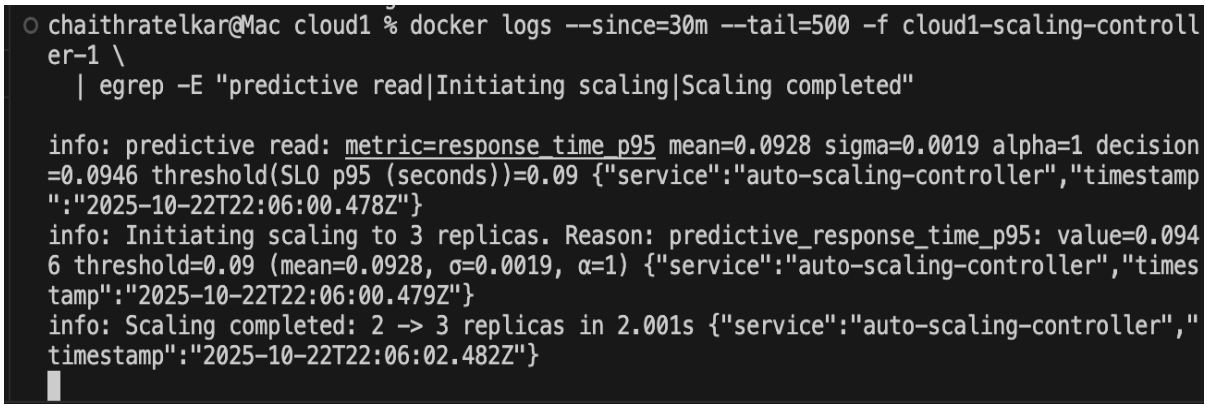

Fig 4.2.8 Controller logs for a predictive scale-up.

4.2.9 Prometheus-Exposed Metrics of the Predictive Auto-Scaling Controller

The figure shows that the Prometheus metrics revealing by the auto-scaling controllers services, which provides all the inclusive runtime informations about the controller's decision making process, scaling operations, and the predictive model outputs. These metrics are getting collected and they monitored by the Prometheus, allowing the real-time visualization and the analysis through Grafana dashboards. The exposed metrics plays the crucial role in the understanding of howthe predictive auto-scaling systems behaves under the different workloaded conditions and how it effectively it maintains service performance.

The key metrics displayed including scaling_decisions_total, which are representing the total number of the scaling decisions made by the controller, categorized by their decision types and reasons, such as predictive or manual scaling actions.scaling_events_total indicates that the count of individual scaling events, including both scale-up and scale-down operations are performed by controller. The metrics current_replicas and target_replicas are represent the currently running replicas and the desired replica count determined by the controller's scaling logic.

Additionally, scaling_latency_seconds records the time taken to the complete each scaling operations, providing valuable insight to the responsiveness and efficiency of the controller. The predictive outputs from the machine learning models are captured using the predictive_mean, predictive_sigma, and predictive_upper, where there is mean

representing the predicted value, sigma (σ) denotes the uncertainty in the prediction, and the upper refers to the computed decisions boundary calculated as (mean + α·σ). The predictive_threshold metrics defines the SLO (Service Level Objective) limit against which their are predicted values is compared to the trigger scaling actions. Lastly, the predictive_wants_scale indicating whether the controller currently intends to the perform of scaling action, where a value of 1 denotes the scale-up intent and 0 represents no scaling required.

Together, these metrics forms the telemetry bridge between the machine learning prediction module and the scaling controller, enabling the Prometheus and Grafana to visualize the predictive trends, scaling frequency, latency, and the stability of the system in real time. This integration provides the transparent and data-driven viewing of how the predictive intelligence is applied to the maintaining optimal system performance through proactive auto-scaling.

```
# HELP scaling_decisions_total Total number of scaling decisions made
# TYPE scaling_decisions_total counter
scaling_decisions_total{decision="scale",reason="predictive_response_time_p95: value=0.0941 threshold=0.08 (mean=0.0928, σ=0.0014, α=1)"} 19
scaling_decisions_total{decision="scale",reason="manual_test"} 2

# HELP scaling_events_total Total number of scaling events
# TYPE scaling_events_total counter
scaling_events_total{event_type="scale_up",target="app"} 19
scaling_events_total{event_type="scale_down",target="app"} 2

# HELP current_replicas Current number of replicas
# TYPE current_replicas gauge
current_replicas 10

# HELP target_replicas Target number of replicas
# TYPE target_replicas gauge
target_replicas 10

# HELP scaling_latency_seconds Time taken to complete scaling operations
# TYPE scaling_latency_seconds histogram
scaling_latency_seconds_bucket{le="0.1"} 0
scaling_latency_seconds_bucket{le="0.5"} 0
scaling_latency_seconds_bucket{le="1"} 0
scaling_latency_seconds_bucket{le="2"} 3
scaling_latency_seconds_bucket{le="5"} 21
scaling_latency_seconds_bucket{le="10"} 21
scaling_latency_seconds_bucket{le="+Inf"} 21
scaling_latency_seconds_sum 42.02199999999999
scaling_latency_seconds_count 21

# HELP predictive_mean Predictive mean
# TYPE predictive_mean gauge
predictive_mean 0.09275403594970703

# HELP predictive_sigma Predictive sigma
# TYPE predictive_sigma gauge
predictive_sigma 0.0018729168176651002

# HELP predictive_upper Predictive upper bound
# TYPE predictive_upper gauge
predictive_upper 0.09462695276737214

# HELP predictive_threshold Predictive threshold
# TYPE predictive_threshold gauge
predictive_threshold 0.08

# HELP predictive_wants_scale Predictive scale desire 0/1
# TYPE predictive_wants_scale gauge
predictive_wants_scale 1
```

Figure 4.2.9 Prometheus-exposed controller metrics showing predictive, decision, and scaling-related telemetry data

4.2.10 LSTM Workload Prediction

Figure 4.2.10, where the LSTM workload forecast results shown above demonstrate the model's ability to learn the sequential temporal patterns and predicting future workloads behavior based on the historical observations. The last five actual values (t−4 to t−0) representing the recently observed response_time_p95 latencies, ranging approximately between 91.88 and 93.06 seconds, indicating a moderate and stable systems load. The predicted next five values (t+1 to t+5) shows a gradual decrease from 91.17 to 90.58 seconds, which indicating the system is expected to experience reduced latency and workloads in the upcoming intervals. This downward trend reflecting the LSTM model's capability to the capture workload dynamics and short-term dependencies effectively from the VMCloud datasets. By identifying such trends early, the controller can proactively prepare for changes in the systems demand and adjusting resources accordingly. This proactive predictions helps in preventing latency violations and maintaining the service stability without the unnecessary scaling actions. The LSTM model, through this functionality, enhancing the predictiveness of the auto-scaling controller by forecasting near-future workload conditions and providing time-ahead insights for the decision-making. This demonstrates how the LSTM contributes to the predictive auto-scaling mechanism by using learned temporal dependencies to make it data-driven, forward-looking scaling decisions.

```
[LSTM] Workload Forecast Analysis:
  Last 5 actual workload values (response_time_p95):
    t-4: 91.8840
    t-3: 93.1301
    t-2: 93.1301
    t-1: 93.0661
    t-0: 93.0661

  Predicted next 5 workload values:
    t+1: 91.1765
    t+2: 90.7248
    t+3: 90.9217
    t+4: 90.6851
    t+5: 90.5854
LSTM workload forecast completed ✅
```

Fig 4.2.10 LSTM-Based Workload Prediction

4.2.11 Random Forest Workload Prediction

The Random Forest workload forecast results are shown in the Figure 4.2.11 which illustrates the model's capability to the perform of short-term workload prediction by learning nonlinear relationships among the system-level features such as CPU utilization, memory usage, and request rate. The last five actual workload values (t−4 to t−0) are representing the recent observed response_time_p95 values, ranging in between 91.88 and 93.06 seconds, indicating stable system performance without the major fluctuations. The predicted next-step workload, with p50 (median) at 93.0661 and upper confidence bounds (p90 and p95) around 93.0703 seconds, shows the model's high precision and low prediction variance. The narrow confidence range showing that the model expects the workloads and response time to remain constant in the upcoming interval, implying that the no scaling event is immediately required. The Random Forest model's strength lies in its ability in capturing complex dependencies between the performance metrics, making it a strong baseline for point-based workload forecasting. The quantile-based outputs (p50, p90, p95) providing uncertainty-aware predictions that helps in assess the range of possible outcomes. This prediction helps the controller to verify the LSTM model's forecast trends and adds the reliability to the overall predictive scaling frameworks. Overall, the Random Forest model enhancing the robustness of the system by supporting accurate, interpretable, and data-driven scaling decisions.

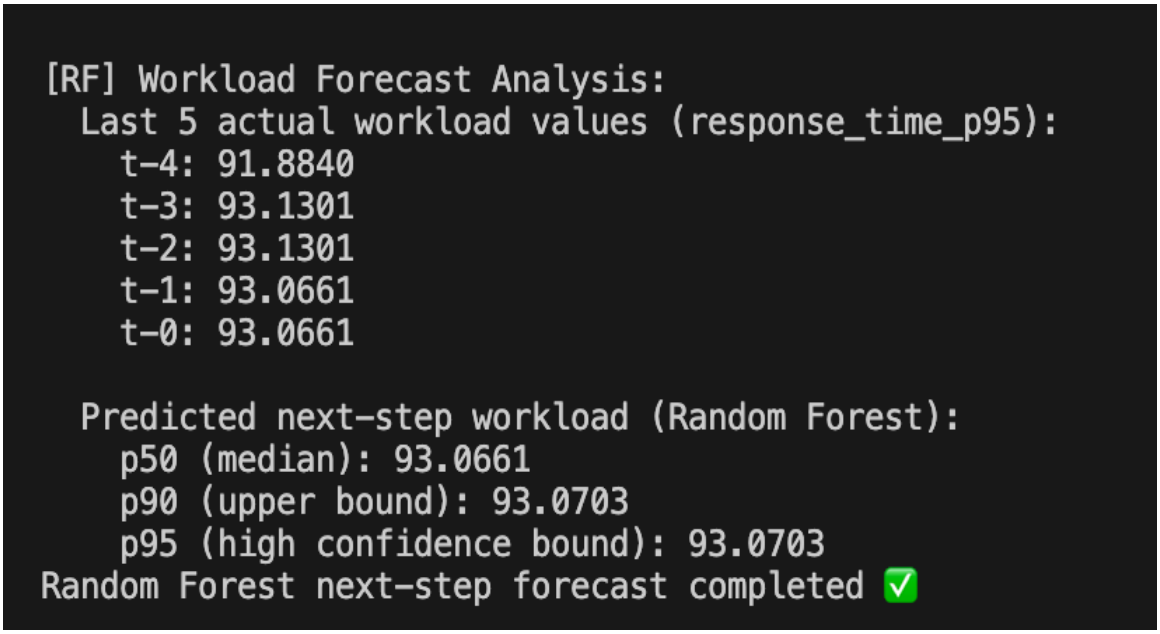

Fig 4.2.11 RF Based Workload Prediction

4.3 Discussion

The discussion explains the efficiency of the predictive auto-scaling framework in maintaining stable system performance and optimizing resource utilization. The controller initiates scale-up actions only when the predictive upper bound, computed as mean + α·σ, exceeds the predefined SLO value, ensuring that scaling decisions are both necessary and data-driven. The Random Forest model provides accurate point-based workload forecasts, contributing stability and precision to the system, while the LSTM model enhances robustness by incorporating uncertainty estimation to manage workload variability effectively. The decision rule using mean + α·σ helps to reduce false scaling triggers and prevents frequent oscillations in replica counts. The inclusion of a cooldown mechanism ensures controlled scaling intervals, avoiding rapid or unnecessary adjustments. Furthermore, the Prometheus metrics and controller logs deliver complete observability, enabling clear traceability from model prediction to scaling execution. Overall, the framework demonstrates a balanced integration of accuracy, adaptability, and interpretability, achieving predictive, stable, and transparent auto-scaling performance.

Table 4.3 Results Summary

Aspect	**Description / Observation**	**Outcome / Result**
Model Performance (MAE / RMSE)	From the evaluation results, the Random Forest (RF) model achieves the lowest MAE (≈0.052) and RMSE (≈0.142), indicating the highest accuracy among all three models. The LSTM model, with MAE ranging between 1.86–2.30 and RMSE between 2.10–2.65, introduces uncertainty estimation capability, while the ARIMA model serves as a classical univariate baseline with MAE ≈2.53 and RMSE ≈3.29.	The Random Forest provides the most accurate point forecasts, whereas the LSTM model contributes predictive uncertainty for risk-aware decision-making.

Predictive Signal Generation	The pipeline generates predictive signals after model training and evaluation, including parameters such as mean and standard deviation (σ), which are stored in artifacts/predictive_signal.json.	The generated predictive signals are utilized by the controller for proactive and data-driven scaling decisions.
Scaling Decision Rule	The controller applies the decision rule formulated as Decision = mean + α·σ, which is compared against the defined SLO_P95_SEC (for example, 0.08 seconds) to determine scaling actions.	Scaling occurs only when the predictive upper bound exceeds the defined SLO, ensuring reliable and necessary scale-up events.
Controller Behavior	The controller logs record key activities such as predictive read, scaling initiation, and scaling completion, confirming the complete scaling cycle.	Verified scaling activity shows the controller successfully scaling from 2 → 3 replicas within approximately 2 seconds.
Prometheus Metrics	Metrics such as scaling_decisions_total, scaling_events_total, predictive_mean, predictive_sigma, and predictive_wants_scale are exposed by the controller and collected by Prometheus.	These metrics enable continuous monitoring, offering complete visibility and traceability from model prediction to scaling execution.
Visualization (Grafana)	Grafana visualizations, including predictive_upper vs threshold, wants_scale vs current_replicas, and scaling event graphs, demonstrate real-time predictive scaling behavior.	The results indicate stable and accurate predictive scaling without oscillations or over-scaling.

LSTM Workload Forecast	The LSTM model predicts the next five workload values, showing a gradual decrease in latency from 91.17 to 90.58, capturing temporal dependencies effectively.	The model accurately forecasts workload trends, supporting proactive latency-aware scaling.
Random Forest Forecast	The Random Forest predicts the next-step workload with narrow confidence intervals (p50 ≈ 93.0661, p95 ≈ 93.0703), maintaining stability in response time predictions.	The model provides reliable short-term forecasts and validates the trend consistency of the LSTM predictions.
Overall Framework Performance	The combined use of LSTM, Random Forest, and ARIMA ensures accurate forecasting, uncertainty management, and interpretability in decision-making.	The framework achieves predictive, adaptive, and stable auto-scaling performance with complete observability and efficiency.

5. Conclusion and Future Work

5.1 Conclusion

The project presents a predictive auto-scaling framework which combines machine learning, system observability, and the intelligent control for maintain optimal performance in cloud environments. Thus the framework integrates a complete data processing pipeline that extracts log and rolling window features and trains three models Random Forest (RF), Long Short-Term Memory (LSTM) with Monte Carlo dropout, and ARIMA to forecast p95 response time. The predictive controller interprets those forecasts, applies a safety factor (α), and the executes scaling actions when the upper bound of the predictions are exceeding the Service Level Objective (SLO_P95_SEC). The system's transparency is enhanced through Prometheus metrics and the structured controller logs, which enable continuous monitoring and the performance traceability. By which adjusting α and SLO_P95_SEC values, the operators can balance the trade-off between reliability (avoiding SLO violations) and efficiency (reducing replica costs). The approach is generalizable to multiple the services and it can integrate seamlessly with the existing monitoring and DevOps ecosystems. Overall, the project demonstrates a practical, reproducible, and data-driven method for the predictive auto-scaling that enhances stability, scalability, and resource utilization.

5.2 Limitations

Although the proposed framework achieves promising results, several limitations exist that define areas for future enhancement. The current implementation focuses on a single objective, optimizing only p95 latency without jointly considering other important metrics such as operational cost, error rate, or CPU utilization. Additionally, the scaling policy parameters (α and SLO) are static and manually configured, lacking adaptive tuning based on feedback or real-time workload sensitivity.

From an MLOps perspective, the system does not yet include automated retraining pipelines**,** feature drift detection, or canary evaluation mechanisms, which are essential for long-term model reliability. Moreover, the dataset used for model training originates from controlled simulations and may not fully capture the burstiness, seasonality, and irregular patterns observed in production environments. Hence, further evaluation with large-scale, real-world traffic datasets would enhance the robustness and generalizability of the results.

5.3 Future Work

Future improvements can be focused on both functional and architectural extensions to increase adaptability and production readiness. The next logical step involves Kubernetes integration, connecting the predictive controller with Horizontal Pod Autoscalers (HPA) or Vertical Pod Autoscalers (VPA) to automate scaling in the containerized deployments. By Introducing hysteresis and the anti-flapping mechanisms which would further stabilize scaling actions and by minimizing rapid oscillations.

Explorations of the advanced deep learning models, such as Temporal Fusion Transformers (TFT), calibrated gradient boosting, and probabilistic neural networks, can improve uncertainty estimation and the prediction accuracy. Additionally, adaptive policies could be developed to dynamically learn α and SLO thresholds using reinforcement learning based on real-time SLO violation cost, user impact, and A/B testing feedback loops.

Extending the systems to multi-cloud and the edge environments would allow scaling decisions in considering cost, latency, and the data locality across the heterogeneous infrastructures. Finally, a stronger MLOps integration is essential in incorporating automated retraining, data and the feature drift monitoring, and the interactive dashboards for the continuous model evaluation and the operational visibility.

Overall, these advancements would transform the current predictive auto-scaling framework into the fully intelligent, self-adaptive, and the production-grade system capable of handling complex and the evolving cloud workloads efficiently.

References

[1] Goggi, Sara, et al. "Semantic Query Analysis from the Global Science Gateway." *Grey Journal (TGJ)* 15.3 (2019).

[2] Pourhabibi, Tahereh, et al. "Fraud detection: A systematic literature review of graph-based anomaly detection approaches." *Decision Support Systems* 133 (2020): 113303.

[3] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. ACM Comput. Surv. 54, 6, Article 130 (July 2022), 37 pages. https://doi.org/10.1145/3460345

[4] Vitali, Monica, et al. "Special issue on co-design of data and computation management in Fog Computing." Future Generation Computer Systems 129 (2022): 423-424.

[5] Finsterbusch, M., Richter, C., Rocha, E., Muller, J. A., & Hanssgen, K. (2013). A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, *16*(2), 1135-1156.

[6] Gezer, A., Warner, G., Wilson, C., & Shrestha, P. (2019). A flow-based approach for Trickbot banking trojan detection. Computers & Security, 84, 179-192.

[7] N. Shone, T. N. Ngoc, V. D. Phai and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, Feb. 2018, doi: 10.1109/TETCI.2017.2772792.

[8] Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., ... & Shen, H. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)*, *51*(5), 1-38.

[9] Alharthi S, Alshamsi A, Alseiari A, Alwarafy A. Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions. *Sensors*. 2024; 24(17):5551. https://doi.org/10.3390/s24175551

[10] Yunda Guo, Jiake Ge, Panfeng Guo, Yunpeng Chai, Tao Li, Mengnan Shi, Yang Tu, and Jian Ouyang. 2024. PASS: Predictive Auto-Scaling System for

Large-scale Enterprise Web Applications. In Proceedings of the ACM Web Conference 2024 (WWW '24). Association for Computing Machinery, New York, NY, USA, 2747–2758. https://doi.org/10.1145/3589334.3645330

[11] Radhika, E. G., & Sadasivam, G. S. (2021). A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment. *Materials Today: Proceedings*, *45*, 2793-2800.

[12] Vinayak Gupta and Srikanta Bedathur. 2022. ProActive: Self-Attentive Temporal Point Process Flows for Activity Sequences. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22). Association for Computing Machinery, New York, NY, USA, 496–504. https://doi.org/10.1145/3534678.3539477

[13] Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, *12*(4), 559-592.

[14] Sun, D., He, H., Yan, H., Gao, S., Liu, X., & Zheng, X. (2021). Lr-Stream: Using latency and resource aware scheduling to improve latency and throughput for streaming applications. *Future Generation Computer Systems*, *114*, 243-258.

[15] A. R. Rathinam, B. S. Vathani, A. Komathi, J. Lenin, B. Bharathi and S. Murugan, "Advances and Predictions in Predictive Auto-Scaling and Maintenance Algorithms for Cloud Computing," *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, Pudukkottai, India, 2023, pp. 395-400, doi: 10.1109/ICACRS58579.2023.10404186.

[16] Flunkert, V., Rebjock, Q., Castellon, J., Callot, L., & Januschowski, T. (2020). A simple and effective predictive resource scaling heuristic for large-scale cloud applications. *arXiv preprint arXiv:2008.01215*.

[17] Ivanovic, M., & Simic, V. (2022). Efficient evolutionary optimization using predictive auto-scaling in containerized environment. *Applied Soft Computing*, *129*, 109610.

[18] Cheng Pan, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. 2019. PRedis: Penalty and Locality Aware Memory Allocation in Redis. In Proceedings of the ACM Symposium on Cloud Computing (SoCC '19). Association for

Computing Machinery, New York, NY, USA, 193–205. https://doi.org/10.1145/3357223.3362729

[19] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. ACM Comput. Surv. 54, 6, Article 130 (July 2022), 37 pages. https://doi.org/10.1145/3460345

[20] Lin, Hai, et al. "A survey on computation offloading modeling for edge computing." *Journal of Network and Computer Applications* 169 (2020): 102781.

[21] Vitali, M., Plebani, P., Bermbach, D., & Elmroth, E. (2022). Special issue on co-design of data and computation management in Fog Computing. *Future Generation Computer Systems*, *129*, 423-424.

[22] Sarma, S. K. (2023). Metaheuristic based auto-scaling for microservices in cloud environment: a new container-aware application scheduling. *International Journal of Pervasive Computing and Communications*, *19*(1), 74-96.

[23] 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.

[24] Reshad, A., & Jammal, M. (2024, December). Optimizing Cloud and IoT Resource Utilization: A proactive, application-agnostic auto-scaling technique using machine learning. In *2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)* (pp. 489-496). IEEE.

[25] W. Z. Khan, Q. -u. -A. Arshad, S. Hakak, M. K. Khan and Saeed-Ur-Rehman, "Trust Management in Social Internet of Things: Architectures, Recent Advancements, and Future Challenges," in *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7768-7788, 15 May15, 2021, doi: 10.1109/JIOT.2020.3039296.

[26] Jadallah, H., & Al Aghbari, Z. (2020). SwapQt: Cloud-based in-memory indexing of dynamic spatial data. *Future Generation Computer Systems*, *106*, 360-373.

[27] Daraghmeh, M., Jararweh, Y., & Agarwal, A. (2025). Leveraging machine learning and feature engineering for optimal data-driven scaling decision in

serverless computing. *Simulation Modelling Practice and Theory*, *140*, 103090.

[28] Murali Krishna N, Vuggam R, Ravuri CN, Devasani RK, loud Resource Forecasting Using LSTM Neural Networks. GJEIIR. 2025;5(4):084.

[29] Manhary FN, Mohamed MH, Farouk M. A scalable machine learning strategy for resource allocation in database. Sci Rep. 2025 Aug 20;15(1):30567. doi: 10.1038/s41598-025-14962-5. PMID: 40835668; PMCID: PMC12368247.

[30] Guruge, P. B., & Priyadarshana, Y. H. P. P. (2025). Time series forecasting-based kubernetes autoscaling using facebook prophet and long short-term memory. *Frontiers in Computer Science*, *7*, 1509165.

[31] Wen, L., Xu, M., Toosi, A. N., & Ye, K. (2024, July). Temposcale: A cloud workloads prediction approach integrating short-term and long-term information. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)* (pp. 183-193). IEEE.

[32] Xu, M., Song, C., Wu, H., Gill, S. S., Ye, K., & Xu, C. (2022). esDNN: deep neural network based multivariate workload prediction in cloud computing environments. *ACM Transactions on Internet Technology (TOIT)*, *22*(3), 1-24.

[33] Hybrid and Advanced Technologies Proceedings of the International Conference on Hybrid and Advanced Technologies (ICHAT 2024), April 26-28, 2024, Ongole, Andhra Pradesh, India (Volume 2). Edited ByS. Prasad Jones Christydass, Nurhayati Nurhayati, S. Kannadhasan