TimeSeries Forecasting for Workload Management in Cloud Infrastructure

Shankha Shubhra Gupta

Registration Number: 2310318

A thesis submitted for the degree of **Master of Science in Computer Networks and Security**

Supervisor: **Dr. Mays Al-Naday**

School of Computer Science and Electronic Engineering

University of Essex

August 2024

Abstract

The research addresses the difficult problem of workload management optimization in cloud computing environments, where the unpredictable and dynamic nature of task and usage demands poses major challenges in optimizing performance and preserving efficiency. This problem is serious because inefficient resource allocation can result in under or over-provisioning, which may hinder performance and possibly violate service level agreements (SLAs), or over-provisioning, which wastes computing resources. In order to overcome this, the study proposes a predictive modeling strategy that makes use of time series forecasting techniques namely LSTM, XGBoost, and ARIMA in order to precisely project future resource requirements and allow for proactive scaling. Despite the fact that every algorithm had distinct pros and cons, XGBoost proved to be the most successful model, with over 90 percent accuracy through feature engineering and hyper parameter tweaking. In cloud computing environments, this offered a dependable way to optimize resource allocation, improve system performance, and guarantee constant service delivery.

Acknowledgement

I want to sincerely thank Dr. Mays Al-Naday, my supervisor, for her unwavering support and direction during this research. She trusted me, which motivated me to do much better than I had thought I was capable of, despite our original modest expectations. Particularly in the beginning, when her knowledge of Google workload cloud literature was crucial in determining the course of my work, Dr. Mays regularly offered priceless insights. She was especially helpful with the NCL setup, which really improved the project's quality and streamlined my process. Her kind advice and support helped me get over every obstacle I had, which is how this accomplishment was made possible.

Contents

1 Introduction

The study focuses on using predictive modeling approaches, particularly time series forecasting, to optimize workload management in cloud computing settings. Effective workload management is essential to preserving system performance and cost-effectiveness in the quickly changing cloud computing environment. In order to make smarter scaling decisions, including whether to provide or decommission resources, predictive modeling is essential for predicting resource demands.[34] Because cloud workloads are dynamic, this proactive strategy helps ensure maximum resource usage and reduces operational costs—a requirement.[50]

1.1 Problem Definition

The very dynamic and unpredictable nature of task needs presents substantial obstacles for effective workload management in cloud computing. When cloud service providers overprovision, they allot more resources than what is required to meet the task at hand. This is sometimes done as a safety precaution to avoid downtime or performance degradation and to accommodate unforeseen demand surges. Although this cautious strategy guarantees dependability, it wastes resources in times of low demand and raises operating expenses because idle capacity must be maintained. On the other hand, under-provisioning refers to the allocation of insufficient resources, which may result from incorrect workload estimates or unexpected, abrupt increases in demand.Due to service-level agreement (SLA) violations and lost revenue opportunities, this leads to performance bottlenecks that cause delays, increased latency, or even outages. These consequences have a bad effect on the user experience, damage the provider's reputation, and result in financial losses.[47] Thus, in cloud computing settings, ensuring maximum performance and cost-efficiency requires striking a balance between over- and under-provisioning.[30]

1.2 Scope

The principal objective is to create a model that can predict these important workload parameters with sufficient accuracy. This makes it possible for cloud providers to decide how best to scale resources up or down in order to effectively manage a range of workloads. Understanding and learning about the behavior of workload data in cloud systems is a major component of this research. As this study's exploratory data analysis (EDA) phase revealed, abrupt spikes and changes are frequently seen in real-world cloud data. These spikes show abrupt shifts in workload requirements, which are typical in circumstances involving cloud computing. Predicting and responding to these sudden shifts is essential to preserving cloud service performance and stability. The initiative intends to improve the system's adaptability and increase its resistance to fluctuations by simulating these erratic patterns.[4][43]

1.3 Value for the Solution

The utilization of predictive models in workload management can yield significant benefits such as optimized resource allocation, decreased operating expenses, and increased performance of cloud services. Cloud providers can avoid the inefficiencies of reactive management by using forecasts on workloads, machines, CPU, and memory use to make proactive scaling decisions. By averting overload and underutilization situations, this proactive strategy not only maximizes resource utilization but also improves cloud service dependability and user experience.[3][32]This study assessed ARIMA, LSTM, and XGBoost, three well-known time series forecasting models. Although LSTM (Long Short-Term Memory networks) is well-known for its ability to capture long-term dependencies in sequential data, and ARIMA is typically used for linear time series data, XGBoost's robust regularization techniques and capacity to handle complex, non-linear relationships[39] made it especially useful for forecasting the complex workload patterns seen in cloud environments. XGBoost surpassed both ARIMA and LSTM by optimizing hyperparameters like learning rate, max depth, and number of trees, proving to be the best model for forecasting cloud resource utilization and assisting with scaling decisions.

1.4 Structure of Work

The thesis is divided into multiple important sections to provide a thorough knowledge of the research. The **Literature Review** looks at current approaches and strategies for workload management and resource demand forecasting in cloud systems, with a particular emphasis on machine learning methodologies. The Google Cluster-Usage Traces dataset[**?**], which offers comprehensive information about jobs, tasks, and resource usage to develop and test forecasting models, is used in the **Methodology** section to describe the data collecting method, model selection, and training techniques.

In order to make sure these models attain the necessary accuracy and dependability for forecasting workload patterns and guiding scaling decisions, the **Evaluation** section evaluates them using a variety of key performance indicators (KPIs). Utilizing data visualization techniques to spot patterns, seasonalities, and anomalies—like abrupt workload spikes—was a crucial component of the study. These insights are crucial for comprehending cloud workload behavior and improving the forecasting models. Hyperparameter tuning was also essential for maximizing model performance, particularly for XGBoost. Through the use of cross-validation techniques and methodical adjustments to parameters like learning rate, max depth, and the number of estimators, the models were strengthened and improved to the point where they could reliably predict dynamic and unpredictable cloud workload data.

2 Literature Review

This chapter will go over the fundamental state-of-the-art information needed to comprehend the research question and value the past and present work in the field. Research and initiatives related to cloud computing workload management are included, along with techniques for enhancing the effectiveness and quality of resource distribution. Furthermore, the chapter will examine several forecasting methods, including time series analysis and machine learning models, that are employed to anticipate cloud resource utilization.

2.1 Workload Management in Cloud Computing

By providing a full range of services—including storage, processing power, and web hosting—through a single, scalable platform, cloud computing has completely changed the IT sector. Due to this paradigm shift, companies of all sizes may now utilize cloud services without having to spend a lot of money on and keep up large private infrastructures. The pay-as-you-go model offered by cloud services is particularly attractive since it allows enterprises to take advantage of the scalability and reliability of cloud environments while also drastically lowering operational costs.[47]

Since more businesses are moving their services and apps to cloud platforms, cloud adoption is growing, making effective workload management more and more crucial. In cloud computing, efficient workload management is critical to maximizing resource allocation, guaranteeing performance, and striking a balance between cost effectiveness. It tackles issues like underprovisioning, which may result in lost opportunities and possible income loss, and overprovisioning, which can lead to resource waste. [9]

[18] provides a thorough analysis of cloud computing resource management, emphasizing the shortcomings of static, traditional policies and the move toward data-driven, machine-learning-based strategies. Numerous resource management activities are covered by it, such as job scheduling, virtual machine consolidation, workload estimate, resource optimization, and energy optimization. The first section of the paper introduces key ideas in cloud computing, including deployment methods, service models, and the use of machine learning in cloud environments. After that, it explores resource management difficulties and groups them according to several factors such as workload forecasting, virtual machine consolidation, resource provisioning, virtual machine location, and temperature control. The article examines current methods for resolving these issues and lists their main advantages and disadvantages.

The cloud computing components while utilizing machine learning are displayed in Figure 1. For effective management of the underlying resources of the cloud architecture, a Resource management system (RMS) collaborates with users and ML prediction components. The ML prediction module consists of data collection, machine learning models, training and validation of ML models, and finally, model deployment for run-time use.

This methodology is essential for workload management because it improves the accuracy of network performance forecasts, enabling more efficient resource distribution.Furthermore, the usefulness of artificial neural networks (ANNs) and auto-regressive integrated moving averages (ARIMA) for traffic prediction in cloud computing is emphasized, highlighting their value in enhancing workload management by offering precise traffic forecasts that facilitate better capacity planning and resource allocation [33]. These cutting-edge methods greatly advance the rapidly developing subject of traffic forecasting and are essential for improving workload management in data center and cloud environments.

In order to anticipate path loss in network data centers, a hybrid approach combining feed-forward artificial neural networks (ANNs), Support Vector Machines (SVMs), and dimensionality reduction approaches is described in the article published in [28].

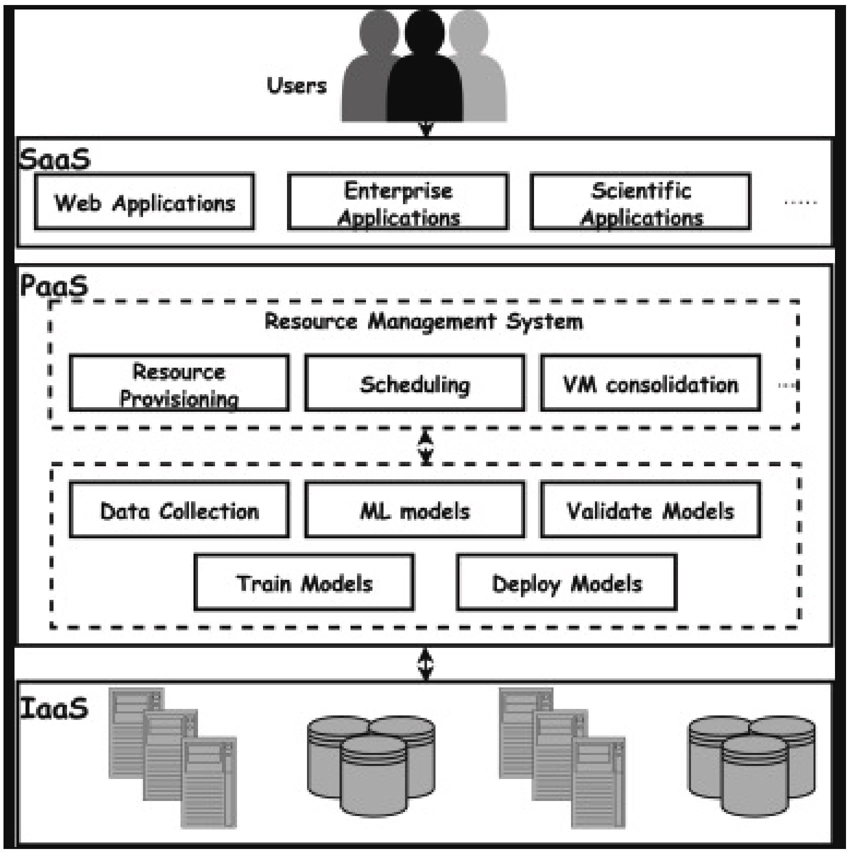


Figure 1: Resource Management System
[18]

2.2 Challenges in Resource Allocation

The challenging of resource allocation in cloud computing is impacted by the varied and dynamic nature of user-submitted workloads. Computational resources are needed for different kinds of workloads, and these needs vary over time (from hourly to yearly variations), making it challenging to effectively estimate and distribute resources.
The under or overuse of resources as a result of uneven Virtual Machine (VM) placement is one of the main problems. Underutilization of resources results in severe resource waste, whilst overloading the system might cause performance issues. This imbalance is frequently brought about by imprecise estimates of resource requirements, which might result from cloud customers' lack of precise data or experience. Because of this, users could overestimate the resources they require and request more than is necessary, which makes the issue of low server usage worse.Furthermore, servers still use a lot of energy even when they are idle. Turning off these servers can result in significant energy savings, but there is a performance trade-off, which emphasizes the need for a careful balance between high performance and energy conservation. The visualisation of under load and overload in Cloud environment is shown in Figure 2. [48][32]

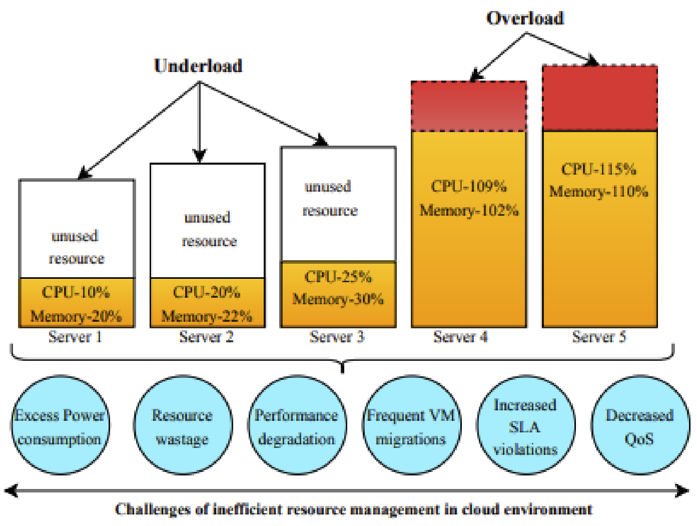


Figure 2: Challenges in Resource Allocation
[32]

Allocating resources like CPU, memory, storage, and network bandwidth to virtualization units like virtual machines or containers is known as resource management in cloud computing. Depending on their main goals, which usually include decreasing work completion time, reducing make-span time, boosting resource efficiency, lowering prices, and optimizing energy use, cloud providers differ in how they manage their resources effectively. These goals are critical; for example, Google discovered that a 0.5-second delay in search results reduces traffic by 20 percent, while Amazon claims a 1 percent revenue loss for every 100ms increase in response delay [8].
Likewise, power consumption accounts for almost 70 percent of all data center running expenses, underscoring the significance of reducing energy use [31]. Traditionally, algorithms and heuristics have been used to solve these problems. Nevertheless, these approaches frequently lack dynamic adaptability to changes in the workload, are not workload-specific, and require workload knowledge in advance of parameter tweaking. Several researchers have resorted to machine learning as a solution to these constraints since it provides a workload-specific method, manages changing workload behaviors, and doesn't require prior workload specialization.

2.3 Workload Prediction Techniques

Numerous studies examined methods for improving surroundings' suitability for applications by employing algorithms that learn from experiences.

In their thorough analysis of the many workload prediction techniques used in cloud computing, Masdari and Khoshnevis [23] stress the significance of precise workload prediction for improving cloud performance, cutting energy use, and guaranteeing Quality of Service (QoS) standards. In their review, they recognize how important workload prediction is to proactive resource management and auto-scaling in cloud data centers (DCs), both of which are necessary for cloud service providers to be scalable. Additionally, they [23] offer a thorough analysis of the benefits, drawbacks, and contributions of the current workload prediction techniques. In addition to highlighting the areas and gaps that require more research, they also emphasize how each design has progressed the profession. Their study is essential for pointing out the advantages and disadvantages of different strategies, providing practitioners and researchers with insightful knowledge that will help them improve cloud resource management by more accurately predicting workloads. Figure 3 gives us an idea about state-of-the-art resource management methodology.

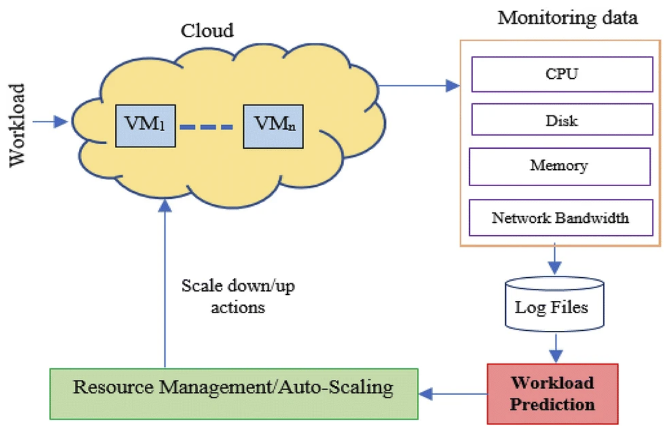


Figure 3: Demonstration of Resource Management using Prediction Techniques [23]

Using a neural network and a self-adaptive differential evolution algorithm, Kumar and Singh [20] provide a unique workload prediction model. The main innovation of the model is its capacity to learn and adjust the best crossover rate and mutation approach during the prediction process. The model performs noticeably better than more conventional prediction techniques, such those that rely on the backpropagation learning algorithm, thanks to its adaptive methodology. The tests show the model's resilience over a range of prediction intervals using benchmark datasets from NASA and the HTTP traces of Saskatchewan servers.

An alternative method is presented by Prevost, Nagothu, and Kelley[29], who combine load demand prediction with models of stochastic state transitions. The goal of this framework is to minimize energy usage while maintaining the necessary performance levels in order to optimize the allocation of cloud resources. The writers concentrate on describing neural networks' and auto-regressive linear prediction algorithms' predictive powers in cloud data center applications. These models' performance is assessed using two different datasets at several look-ahead times, demonstrating how well they can predict load needs in a range of scenarios. A viable path toward more effective resource management in cloud systems is the combination of stochastic models with predictive algorithms.

The dynamic issues of resource provisioning in cloud systems are examined by Amiri and Mohammad-Khanli [4], especially when dealing with the limitations of abrupt demand swings. The authors stress the need for a delicate balance between under-provisioning, which runs the risk of violating Service Level Agreements (SLAs) and lowering Quality of Service (QoS), and over-provisioning, which results in energy waste and higher costs. They stress that precise workload and performance forecasts are essential for efficient resource supply. The authors create a taxonomy of application prediction models by analyzing cutting-edge prediction techniques and classifying them according to their primary traits and difficulties.

Ahamed, Khemakhem, and Eassa [3] point out that previous studies in this field frequently depend on individual data sources and undervalue the significance of data consistency, which is necessary to generate objective and precise analysis. This mistake has the potential to seriously impair DL models' ability to anticipate workloads. The ability of several DL models, including Recurrent Neural Networks (RNN), Multilayer Perceptrons (MLP), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN), to take advantage of the time-series features of real-world workloads is examined technically by the authors in order to close this gap.

A novel framework named LoadDynamics is introduced by Jayakumar, Lee, and Kim [14]. Its purpose is to provide very accurate workload forecasts for a range of applications. In contrast to numerous other models that are customized for certain workload types, LoadDynamics is a general framework that can adjust to various workloads by automatically fine-tuning its internal settings. Maintaining high prediction accuracy in a variety of settings, such as web services, scientific computing, public cloud applications, and data center workloads, requires this flexibility.Long Short-Term Memory (LSTM) models, which are especially useful for time-series prediction because they can capture long-term dependencies within data, are the foundation of LoadDynamics.

2.4 Timeseries Forecasting

Time-series patterns in workloads found in contemporary data centers require training prediction models on historical data under the presumption that trends in the future would reflect past actions. Unfortunately, new patterns frequently emerge as a result of the highly nonlinear nature of workload variations in data centers, making it more difficult for the model to learn and forecast with accuracy. An ensemble approach is commonly used to address this difficulty because no one model can handle all sorts of time-series prediction data in an effective manner[42]. However, the fixed collection of predictors that many ensemble models rely on, restricts their capacity to adjust to evolving patterns in time-series data.

Because time series models are based on widely available historical sequences of observations from secondary sources, they can be useful in some situations, especially when forecasting. Time series modeling focuses on examining the statistical dependencies between these subsequent observations. Time series models forecast future values based on historical trends of the research variable itself, as opposed to models that call for explanatory variables.This method is helpful for two main reasons: first, it can be difficult to assess cause-and-effect links, or the underlying system may not be well known; second, the main objective might be to forecast outcomes rather than comprehend their causes. Furthermore, gathering long-term data on explanatory variables can be difficult or impossible when it comes to causal factors. Time series models provide forecasters with a useful solution in these kinds of situations by allowing them to make forecasts without requiring intricate causal information.[26]

[15] presents an ensemble method that combines different prediction models to increase accuracy in estimating CPU load in cloud environments. This ensemble strategy is made to adjust to changes in resource availability and time because the optimal model for estimating CPU load can change over time. It consists of two layers: the first layer adds and removes high-performing prediction models to optimize the prediction models to utilize. In addition to providing feedback to help the first layer get even better, the second layer integrates the output from these models to produce a final forecast. Real CPU load data from a private cloud was used to test the method, and the results demonstrate that it works better than any one model.

Efficient resource management in cloud computing is highly dependent on precise forecasts of resource consumption, including CPU and memory. Time series forecasting techniques have shown to be useful in this context, especially as cloud workloads frequently have dynamic and time-varying properties. Cloud systems can more effectively manage capacity planning and reduce the hazards of over- and under-provisioning by forecasting future resource consumption. To increase prediction accuracy, forecasting techniques include strategies like mixing linear and nonlinear time series forecasting models. For example, a hybrid ARIMA-ANN model combines the best features of the Artificial Neural Network (ANN) for modeling nonlinear characteristics with the ARIMA model for capturing linear patterns in resource utilization data

Additionally, the utilization of ensemble-based forecasting methodologies has demonstrated potential in augmenting prediction accuracy through the combination of different models. Combining the strengths of ARIMA, XGBoost, and LSTM models, for instance, makes use of each model's unique advantages: LSTM can model temporal relationships in time series data, while ARIMA can capture linear trends. XGBoost can handle structured data. [34][6]

By utilizing temporal relationships to identify trends and patterns, time series forecasting with machine learning predicts future values based on previously observed data points. When operations optimization depends on comprehending and controlling future workloads, this strategy is essential. For example, in the context of warehouse management, time series forecasting was used to anticipate workloads across different workstations, as demonstrated in a research carried out on an automotive spare part warehouse in Turkey. The study showed how precise workload prediction might improve employee assignment and overall warehouse performance, avoiding delays and disturbances, by applying both traditional time series approaches and Machine Learning models like XGBoost.[17]

Similar to this, in cloud computing, predictive analysis of server workloads depends on time series forecasting utilizing machine learning models like Long Short-Term Memory (LSTM). Cloud providers rely on these models to allocate resources efficiently, preventing service outages and lowering energy use, as was covered in a different study. The study demonstrated how cloud enterprises could predict future trends and modify their resource management strategies accordingly by utilizing the predictive capabilities of LSTM. This highlights the significance of Machine Learning in both cloud computing and warehouse environments.[45]

3 Methodology

3.1 Setup

My supervisor, **Dr. Mays Al-Naday**, gave me a specialized machine that is a component of the National Computer Laboratory's (NCL) infrastructure to evaluate my project. This particular machine was set aside specifically to help my project's computing requirements.

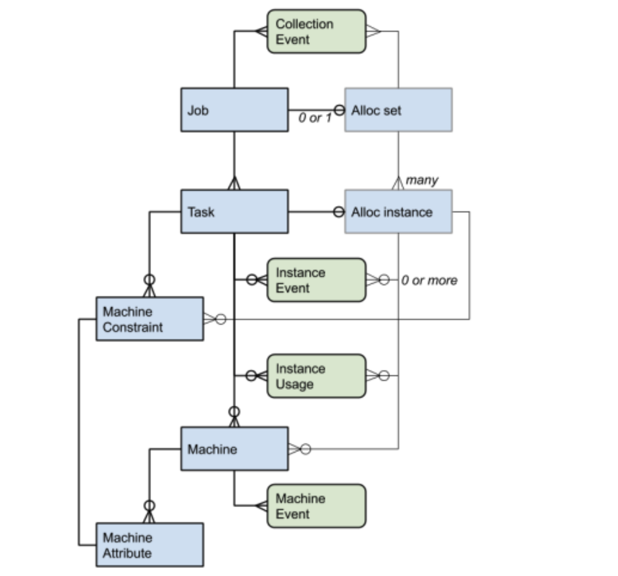
The machine is equipped with the following hardware and OS:
Operating System: Xubuntu 22.04
CPU: Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz, providing sufficient computational power for data-intensive tasks.
Memory: 16 GB of RAM, ensuring smooth operation even with large datasets.
Network Interface: Two Gigabit Ethernet interfaces (Broadcom NetXtreme II BCM5716), allowing for fast data transfer.
Once I had access, I used common Linux commands to move my dataset from my desktop to the local PC. I then installed the required software for my data analysis and model building which includes Python and Jupyter Notebook.Specifically, Python Version 3.8.10. Using the Xubuntu package manager, the installation was simple and finished.I utilized the SSH protocol to get remote access to the computer.By doing this, I was able to safely tunnel the ports required by Jupyter Notebook and have remote access to its resources from my home computer.With all required tools and resources accessible remotely, this configuration allowed me to conduct my research in a reliable and effective environment. My project's effective completion was made possible by the NCL infrastructure's high-performance computer and secure access.

3.2 Dataset Overview

The Google Cluster-Usage Traces [16](version 3) dataset is a thorough set of data that captures the operational aspects of a Google Borg cluster over a given period. Google's massive cluster management system, known as Borg, is made to manage a wide range of workloads on thousands of machines. To be precise , I have extracted almost 31 days of data from the Google Cluster-Usage Traces, which provides a comprehensive view of the operations within a large-scale cloud infrastructure.This data captures various aspects of resource utilization, task scheduling, and machine events across thousands of servers managed by Google's Borg system. This dataset is a great fit for the goals of this project since it is especially useful for studies on distributed systems, resource management, and workload optimization.
The trace data is arranged into several tables, each of which shows similar and differenet kind of events or cluster entities as compared to each other. These tables include CollectionEvents, InstanceEvents, MachineEvents, MachineAttributes, and InstanceUsage. The data has undergone a number of obfuscation techniques, including scaling and normalizing, to preserve privacy and confidentiality and also making it easy for the researcher or developer to articulate. This guarantees the randomization of sensitive data while permitting insightful analysis. Memory and CPU usage are standardized. For instance, memory is rescaled in accordance with the maximum memory observed among the computers, while CPU is measured in Google Compute Units (GCUs).
Figure 4 gives us a breif idea about the entity relationship model of the google-cluster-usage trace with green boxes being the subsets and blue being the features.[16] A distinct 64-bit identifier is issued to every machine, job, and task in the dataset, facilitating their tracking throughout the trace. This covers all state changes that occur throughout the trace period, both the first and later ones.[16]

Specifically, the **Instance Usage Table of the Google cluster** traces provided the data required in this study. This table provides information on how resources are distributed and used over time by tracking specific resource use metrics for each instance (task or unit of work) operating inside the cluster.Careful data management was necessary while exporting huge datasets from BigQuery[2], making sure that all pertinent fields were included and that no information was omitted to optimize processing. [16]

Google's public dataset repository was used to obtain the Google Cluster-Usage Traces. I used Google BigQuery[2], a fully managed data warehouse service that enables effective querying of enormous datasets, to extract the exact data needed for this study. Relevant data fields that capture different elements of resource utilization, such as CPU and memory usage across predetermined time periods, were retrieved



An entity-relationship model for the objects (blue) and events (green) found in the traces.

Figure 4: Entity Relationship Model
[16][10]

by querying the Instance Usage Table. But instead of utilizing the complete dataset, I extracted the pertinent columns that were required to meet the study's goals. To concentrate on the metrics essential for predictive modeling, a subset of the Instance Usage Table's columns was selected. The aggregate resource utilization of these instances at the collection or task level throughout the designated time period is really reflected in each column, even though the dataset is primarily focused on individual instances. To evaluate the data and comprehend its significance for resource management in a cloud datacenter setting, this distinction is essential.
Therefore, the retrieved data was downloaded in CSV format for additional processing and examination.The start time and end time timestamps in the dataset were initially expressed in microseconds. During preprocessing, these were changed to seconds to comply with the analysis's 300-second interval requirement. The dataset's columns, which each represent a 300-second interval and record the following crucial characteristics: Time Intervals: start time and end time show the span of time, measured in microseconds from the trace's beginning, for each data point.
Identifiers: Within the cluster, collections, instances, and machines are uniquely identified by collection id, instance index, and machine id. This allows resources to be tracked between various system components.
Resource Usage Metrics: A comprehensive perspective of the computational resources used by each instance over the designated interval is provided by the average usage.cpus, average usage.memory, maximum usage.cpus, maximum usage.memory, and other related columns.[16]

A task can be accomplished by several instances cooperating to complete a collection or job. The overall resource consumption of the entire collection is represented by the sum of the metrics, which include CPU, memory, and other resource allocations, at each 300-second interval.The unique count of collection IDs and machine IDs was also taken into consideration in order to more accurately depict the distribution of workloads throughout the cluster. These distinct counts provide information about the composition and volume of the workload being handled, as well as how resources are distributed among various machines and collections. To make sure a collection operates effectively, extra instances or instances with a larger capacity may be assigned to it if it routinely consumes more CPU or memory than other collections. Depending on their total resource demands, machines may be dynamically assigned to collections or reassigned from existing ones. To maximize system performance, collections with high utilization might be given to more powerful computers, while collections with low usage might be assigned to less powerful hardware.

Furthermore, the methodology incorporates the non-unique collection counts as a crucial parameter in my research since I understood the importance of resource usage at the collection level. A more sophisticated knowledge of resource demand patterns is made possible by these counts, which offer insights into the frequency and distribution of collections across time.We may more accurately predict resource needs for recurring tasks and assign resources to those collections by keeping track of non-unique collection counts. This measure provides a thorough understanding of how resources are used throughout the datacenter, complementing other aggregated consumption metrics.

3.3 Feature Engineering and Interval Aggregation

The extracted dataset was loaded into a Pandas DataFrame.Pandas is a strong Python package for handling and analyzing data. It offers data structures like DataFrames, which make handling structured data simple and make operations like filtering, combining, and aggregating datasets possible. Start time, end time, collection id, machine id, and several CPU and memory utilization indicators were among the columns in this dataset.
The start time and end time columns were converted from their original microsecond units to seconds. To align the data with the 300-second interval needed for analysis, this conversion was necessary.
In order to preserve the proper temporal order—which is essential for precise time-series analysis—the data were sorted by start time using Pandas sorting functionality. Sorting makes sure that the order of events is maintained, which makes it easier to aggregate and analyze the data later.Moreover, time-based sorting facilitates efficient feature engineering. The right temporal order enables time-based characteristics, such as moving averages or lag values, to accurately capture temporal interdependence. For example, only if the intervals are processed in the right order would a feature that shows the average CPU consumption over the last five intervals have any value.
Additionally, NumPy, another essential Python module, was utilized to handle arrays and carry out numerical calculations in the background.Pandas was used in this examination to indirectly use NumPy for a number of computations, including rolling averages and the summation of CPU and memory utilization over time.

The process of aggregating the data into 300-second intervals made it possible to analyze trends in resource utilization across reliable time frames.For each interval, several metrics were calculated:
Unique Collection IDs: The count of distinct collection IDs within the interval.
Non-Unique Collection IDs: The count of collection IDs with multiple instances within the interval.
Unique Machine IDs: The count of distinct machine IDs within the interval.
Total Average CPU usage: The average of all the CPU consumption over the interval for all instances linked to each collection. This is the total of the average CPU consumption throughout the course of those 300 seconds for each instance under each collection.
Total Average Memory usage: The average memory consumption throughout the course of the interval for all instances linked to each collection. The average memory use of each instance under each collection over the interval is combined into this measure.
Total Maximum CPU Usage: The total of all instances' maximum CPU usage numbers for each collection throughout the interval. This records the total peak CPU consumption that was seen throughout the designated time period for every instance inside each collection.
Total Maximum Memory Usage: The total of all instances' maximum memory usage values for each collection throughout the interval. This represents the overall peak memory use that was seen during that time for every instance under every collection.

Time-Based Feature Extraction: Predictive modeling features were derived from the extracted measurements. To further capture temporal connections, time-based indices like lagged values were developed. Models can better comprehend and forecast future patterns by utilizing lag characteristics, which integrate historical data sets based on previous behavior.
Target variables: For the purposes of predictive modeling, any calculated metric (such as the total of the average CPU utilization or the unique collection IDs) was considered a target variable. By using these characteristics to predict values for upcoming 300-second intervals, proactive resource management was made possible.

3.4 Exploratory Data Analysis (EDA)

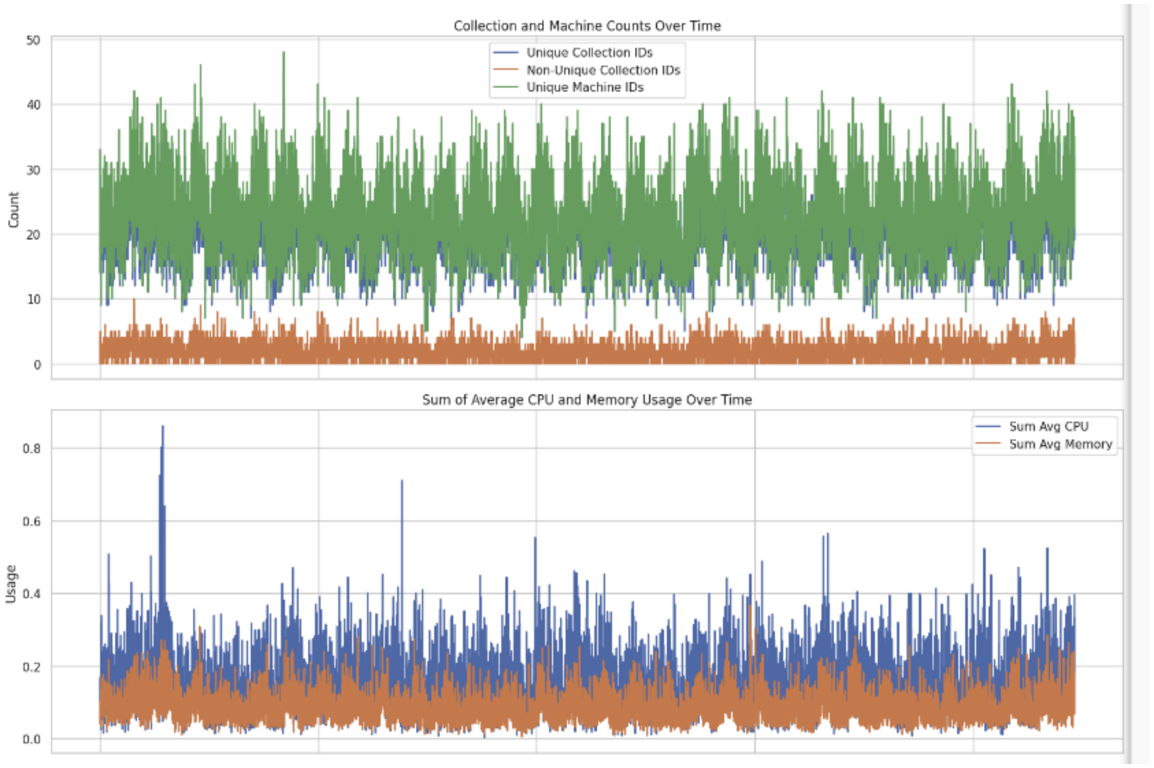


Figure 5: Linechart 1

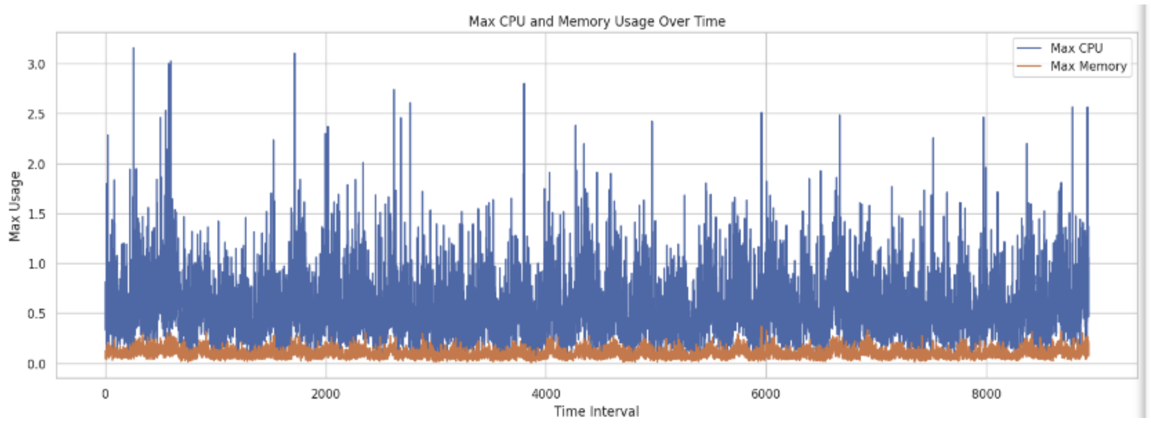


Figure 6: Linechart 2

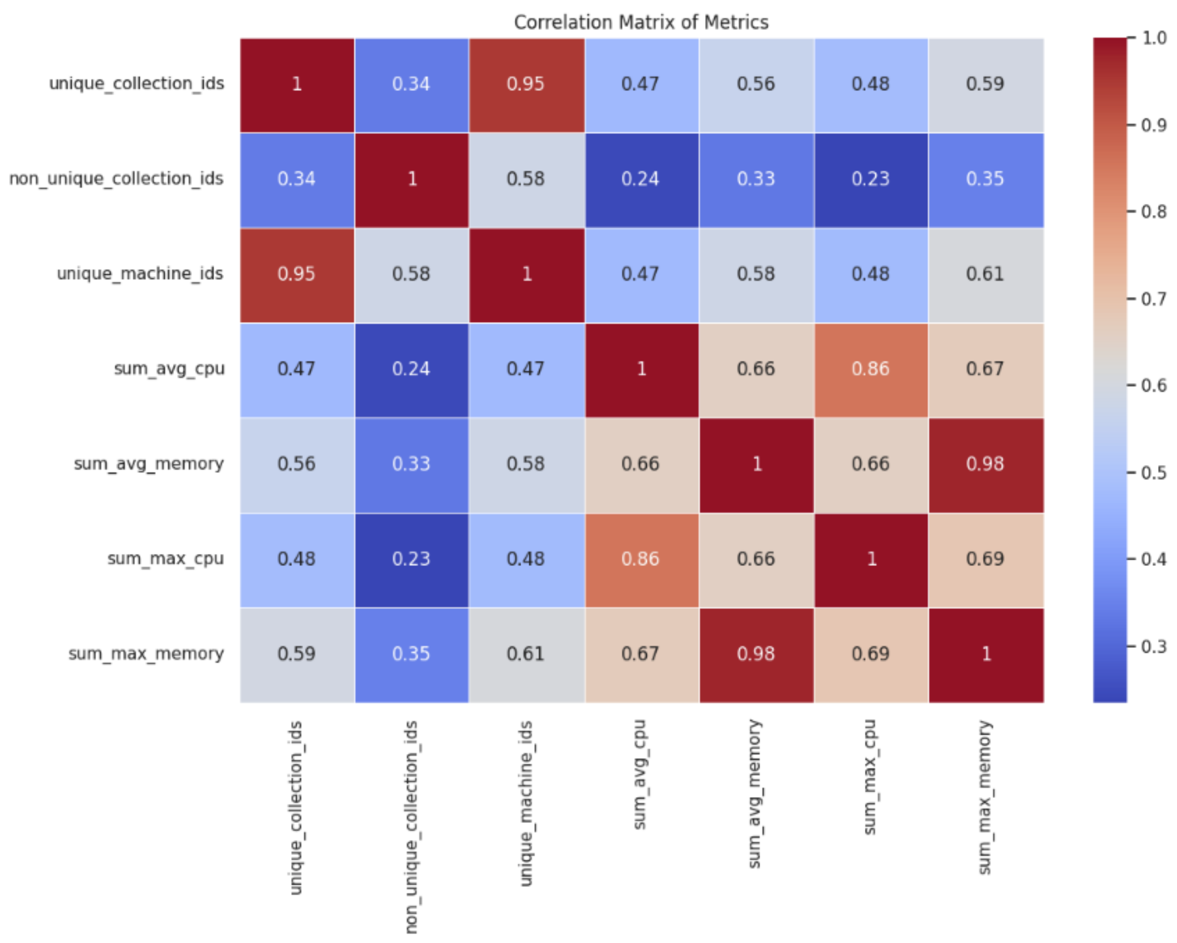


Figure 7: Correlation Matrix for EDA

To learn more about the distributions, trends, and connections between the computed metrics, EDA was used. To visually inspect the data, time-series charts and correlation matrices were used. Strong correlations between variables can result in redundant data. By selecting features based on an understanding of these linkages, you can simplify models without sacrificing predictive power by combining or removing features. Figure 5 and 6 shows the Time-series plots of different features that would be fed to the Prediction Algorithm's done using Feature Engineering and Interval Aggregation. These charts can be addressed as the ground truth of the whole Dataset used in this reaserch. To be specific , these figures show how various counts and usages vary over time. Taking into account that the usages are already present within the range of (0,1) in the original dataset to prevent data leakage.[16]Plotted over time, with time on the x-axis and measured values (counts) on the y-axis, each line indicates the count of a distinct category. Figure 7 shows the correlation matrix between different features to be used in modelling. The correlation matrix give us an insight about the relationships between multiple variables/features which can guide in feature selection, model building and interpretation of results in data analysis.

Augmented Dickey-Fuller (ADF) Test [26]: To verify stationarity, the ADF test was run on each of the important measures. Table 1 shows the outcomes for every metric:

Metric	**ADF Test Statistic**	**p-value**	**Stationarity**
Unique Collection IDs	-8.471	1.47e-13	Stationary
Non-Unique Collection IDs	-8.710	3.61e-14	Stationary
Unique Machine IDs	-8.446	1.70e-13	Stationary
Sum of Average CPU Usage	-8.803	2.09e-14	Stationary
Sum of Average Memory Usage	-7.960	2.96e-12	Stationary
Sum of Maximum CPU Usage	-7.934	3.46e-12	Stationary
Sum of Maximum Memory Usage	-7.722	1.18e-11	Stationary

Table 1: ADF Test Results

The data needs to be stationary, or have statistical characteristics that do not vary over time, for time-series models such as ARIMA.
All measurements' p-values were significantly below the 0.05 cutoff from the ADF test, indicating that the data is stationary, but it has limitations especially when there are trends, excessive noise levels, or structural breaks in the data.[1] It is crucial to visually inspect the data, as done above, even if the ADF test indicates that the data is steady (as indicated by a low p-value). The observed spikes and volatility we see may be a sign of non-stationarity, which could be caused by structural fractures or variations in variance over time that the ADF test may not have fully detected.

Next, an alternative method was used to further verify stationarity of the data .Bartlett's test or Levene's test can be used to test the state of variances.[24][38]
According to the findings of the Bartlett and Levene tests, there is a good chance that the variance in your time series data is uneven, or that it does not vary with time.

- **Bartlett's Test p-value:** 1.57×10^{-192} (extremely small)
- **Levene's Test p-value:** 2.94×10^{-34} (also extremely small)

The data shows asymmetric, which means that the variance is not constant over time, according to the findings of the Bartlett and Levene tests. This is a form of non-stationarity.

Data Normalization and Scaling: No further normalization or scaling procedures were needed because the data had already undergone obfuscation, which included normalization and scaling, to protect privacy. As-is metrics were available for modeling.

3.5 Modelling and Development

To accurately predict the temporal dynamics in time series forecasting, the right model must be selected. In this talk, we look at three different models: Long Short-Term Memory (LSTM), AutoRegressive Integrated Moving Average (ARIMA), and Extreme Gradient Boosting (XGBoost). Each of these models has advantages of its own and is intended for a specific type of time series data and forecasting assignment. The supervised learning model XGBoost is ideally suited for forecasting resource usage patterns in cloud environments because it excels at capturing intricate, non-linear correlations in sizable, feature-rich datasets. Another supervised model, long-term dependencies in sequential data are best learned by LSTM, which is ideal for reliably predicting workload trends over time. A statistical model called ARIMA manages time series seasonality and linear trends well.We intend to take use of each model's distinct advantages and offer a thorough evaluation of forecasting techniques for cloud workload management by applying these models independently.

3.5.1 Long Short-Term Memory (LSTM)

Recurrent neural networks (RNNs) of the Long Short-Term Memory (LSTM) variety were created expressly to get around some of the drawbacks of more conventional RNNs, most notably the vanishing gradient issue[27]. Time series forecasting problems where the associations between observations cover lengthy periods of time are a good fit for lengthy Short-Term Memory (LSTM) models because of their exceptional ability to learn and recall long-term dependencies in sequential data.
Data sequences with dependencies over time can be handled by LSTM. Without actively changing the data to keep it stationary, LSTMs can discover and identify patterns in data, including trends and seasonality. Because of their architecture, which includes memory cells and gates, lengthy sequences of information may be retained by LSTMs, which makes them resistant to non-stationary data.[34]

LSTM Architecture
Memory cells and gates are essential components of LSTM networks because they manage information flow and allow the model to preserve long-term dependencies throughout time steps. Information is stored in the memory cells, enabling the network to maintain crucial data over time. This process is controlled by the input, forget, and output gates. Whereas the input gate chooses what new data should be supplied, the forget gate chooses which information should be removed from the memory cell. The data that should be sent out of the memory cell is managed by the output gate.[44]

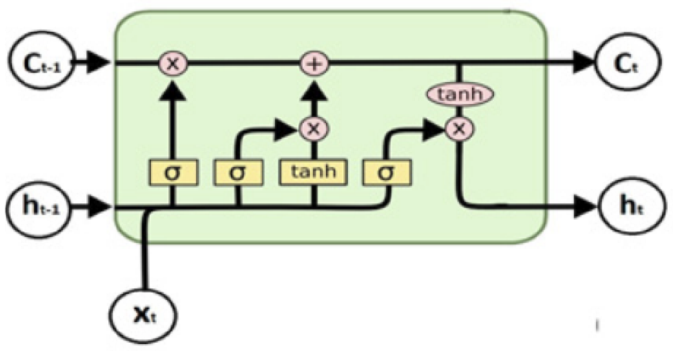


Figure 8: LSTM architecture
[44]

The sigmoid function, which determines how much information is kept or discarded, powers these gates and produces values between 0 and 1. LSTM models analyze data arranged into sequences of a certain length, referred to as sequence length, in the context of time series forecasting. For instance, the model forecasts the subsequent value based on the preceding 10 observations when the series length is 10. LSTM networks can contain numerous layers, each with a different amount of memory cells (units), to capture complicated temporal patterns. Four major hyperparameters affect the model's performance and capacity for generalization: batch size, epochs, sequence length, and number of units.

Several key libraries were used to implement the LSTM model:-
TensorFlow: An open-source machine learning frameworkused to create and train neural networks.
Keras: A high-level TensorFlow-based API that makes deep learning model development easier.
sequential: A Keras model that permits the consecutive addition of layers to create a neural network.
Dense: The fully connected neural network layer that generates the LSTM model's output.
MinMaxScaler: A function that scales characteristics to a given range, usually between 0 and 1, in order to standardize data.

LSTM Hyperparameters
Sequence Length: Number of past observations considered for making predictions.
Number of LSTM Units: Number of memory cells in each LSTM layer.
Batch Size and Epochs: Control the training process.

3.5.2 AutoRegressive Integrated Moving Average (ARIMA)

A traditional statistical model called ARIMA is employed in the analysis and prediction of time series data. It works especially well with data that can be made stationary through differencing and show temporal dependencies. Three components are combined in ARIMA models to represent linear correlations in time series data: AutoRegression (AR), Integration (I), and Moving Average (MA). The autoregressive order, or p, indicates how many lag observations are included in the model.
By using differencing, or the d parameter in the (p,d,q) order, to eliminate trends and seasonality, ARIMA models naturally attempt to keep the data stationary. However, further actions could be required to completely address these problems if the data contains complicated non-stationarities, such as non-constant variance.[12][33]
The **Box-Cox** Transformation was used to stabilize the variance and make the data more normally distributed.Once predictions have been produced, the data can be returned to its original scale by using the inv boxcox function to reverse the Box-Cox transformation.This helped us improve the accuracy and results of the ARIMA forecast a bit.[36]
Libraries used were :
ststsmodels : A Python package for econometrics and statistical modeling that offers tools for data exploration, statistical testing, and model estimation and testing.
ARIMA Model Components:
The link between the current observation and its prior values is captured by autoregression (AR).
Integration (I): To make the time series stationary, it must be differentiated.
When a moving average model is applied to lag observations, the link between an observation and the residual error is modeled using the moving average (MA) technique.
The particular parameters (p, d, and q) used in the fitting of an ARIMA model are selected according to the properties of the data. To render the series stable, an ARIMA(5,2,0) model, for instance, employs five lag observations (AR part), applies differencing twice (I part), and omits the moving average component (MA part). To capture complicated patterns in the data, more sophisticated models may include extra hyperparameters, like seasonal components.

ARIMA Hyperparameters:
p (Lag Order): Number of lag observations included in the model.
d (Degree of Differencing): Number of times the data is differenced to achieve stationarity.
q (Moving Average Order): Size of the moving average window.

3.5.3 Extreme Gradient Boosting (XGBoost)

A powerful machine learning method called XGBoost is built on the gradient boosting architecture. It is well-known for its effectiveness, adaptability, and strong predictive potential and is frequently employed with structured (tabular) data. In order to create a strong model that can identify intricate patterns in the data, XGBoost builds an ensemble of decision trees, each of which fixes the mistakes made by the trees before it.[5]
Decision trees, the foundation of this model, divide the data according to feature space thresholds. Since trends or seasonality have no effect on these divides, the algorithm can somewhat automatically handle non-stationary data. Explicitly including trend or seasonality information can still boost performance, though.

When employing XGBoost for time series forecasting, features extracted from the data—such as the sum of average CPU usage, maximum memory consumption, and the number of unique machine IDs—are used to predict future values, like the number of unique collection IDs. XGBoost operates within a Gradient Boosting Framework by continuously adding new decision trees to the model, each tree aiming to correct the residual errors of the previous one. This iterative process uses gradient descent to minimize a Loss Function, which measures the difference between the model's predictions and the actual values. The gradients of this loss function guide the model to make better predictions in subsequent iterations.[49]
In summary, XGBoost is especially well-suited for forecasting tasks involving rich, time-series datasets because of its robust regularization approaches and capacity to capture complicated, non-linear relationships among features. These components work together to prevent overfitting and guarantee that the model can accurately forecast future trends based on existing data, maintaining excellent generalization to new data.

Libraries used :
XGB : Employs access to a number of classes and functions that let you create, train, and use gradient boosting-based machine learning models. Tools for managing missing values, applying regularization to avoid overfitting, and using parallel processing to expedite calculations are all included in the package. Regression tasks are performed using the **XGBRegressor** class from the xgboost package. The XGBRegressor function from the xgboost package is used to generate the model in this project.[19][39]

XGBoost Hyperparameters:
Learning Rate: Controls the step size during gradient descent.
Max Depth: Maximum depth of each tree, influencing the model's complexity.
Number of Trees: Number of trees in the ensemble.
Regularization Parameters: Include terms like alpha and lambda to penalize overly complex models.

3.6 Scaling Decisons for Cloud

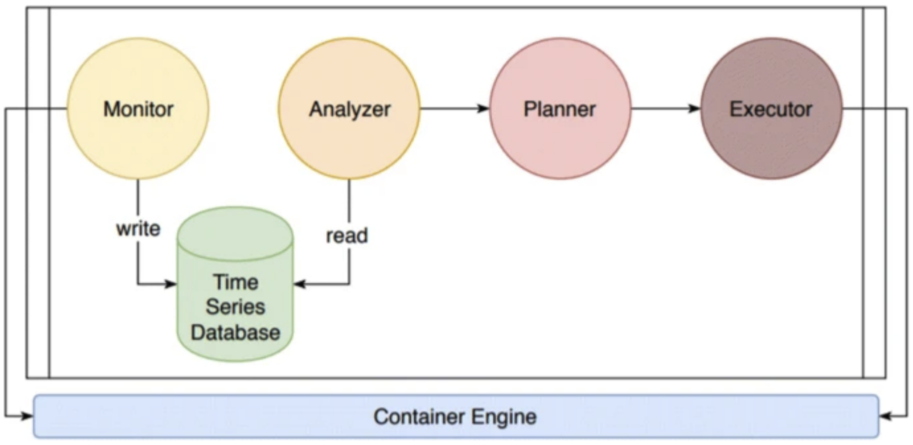


Figure 9: Auto-Scaler Architecture
[13]

To optimize cost and performance, cloud resource management requires the ability to dynamically scale resources in response to shifting demands. The technique employed here to forecast significant indicators pertaining to cloud resources is predictive modeling. The fundamental idea is to forecast future values of important cloud metrics, such as CPU and memory consumption and the number of unique machine or collection IDs. Reactive approaches, which ask for additional resources only when the system hits a certain load threshold, would not be adequate in a cloud computing environment because of resource allocation delays. Predicting future requests and proactively allocating resources are required to solve this and prevent system overload.

The adaptive prediction approach presented by [25] combines many time series forecasting models and employs genetic algorithms to optimize model selection and continuously adjust to new data. This ensures efficient resource allocation even in the case of insufficient historical data. Complex auto-scaling techniques are needed for cloud applications that are based on containers. In order to reduce delays and improve performance, the article by [13] presents a proactive machine learning-based auto-scaling solution for Docker containers. Figure 9 shows the auto-scale architecture using the container engine. It does this by utilizing LSTM networks to forecast future workloads and assure timely provisioning and de-provisioning of resources. In order to avoid frequent scaling oscillations, this method also includes a progressively declining strategy. The results show significant gains in prediction accuracy and cost-efficiency.

Predicting future values of critical cloud metrics, like CPU and memory consumption and the quantity of unique machine or collection IDs, is the main premise of our project.

4 Evaluation

4.1 KPI Metrics

The performance criteria you use for time series forecasting can have a big impact on how you assess your model. The most suitable metrics frequently rely on the particulars of your data and the objectives of your forecasts. The following are a few of the most widely used metrics in time series forecasting[3]:

The mean absolute error, or (MAE), calculates the average error magnitude between the actual and anticipated values without taking the direction of the errors into account. It is simple and has the same units as the information.
The average of the squared discrepancies between the expected and actual values is determined using the **Mean Squared Error (MSE)** method. It's helpful for detecting significant deviations because it penalizes bigger mistakes more severely than smaller ones.
The error metric in the same units as the data is provided by the **Root Mean Squared Error (RMSE)**, which is the square root of the MSE. It is frequently used to evaluate the overall performance of the model and highlights big inaccuracies.
The difference between the expected and actual workload values is measured by the **Mean Absolute Percentage Error (MAPE)**, which is based on the findings that are displayed. Less resources are misallocated when the MAPE is smaller, which is correlated with improved prediction accuracy.

Due to its emphasis on variance explained rather than predictive accuracy, R-squared (R^2) is unsuitable for time series forecasting. Because it ignores temporal relationships, it might not accurately represent projected performance on future data. Elevated R2 values may suggest overfitting since they gauge the fit on past data without guaranteeing future precision. Since predictive accuracy metrics quantify prediction errors rather than the percentage of variance explained, they are more pertinent for assessing forecast performance. Examples of these metrics are MAE, RMSE, and MAPE.[11]

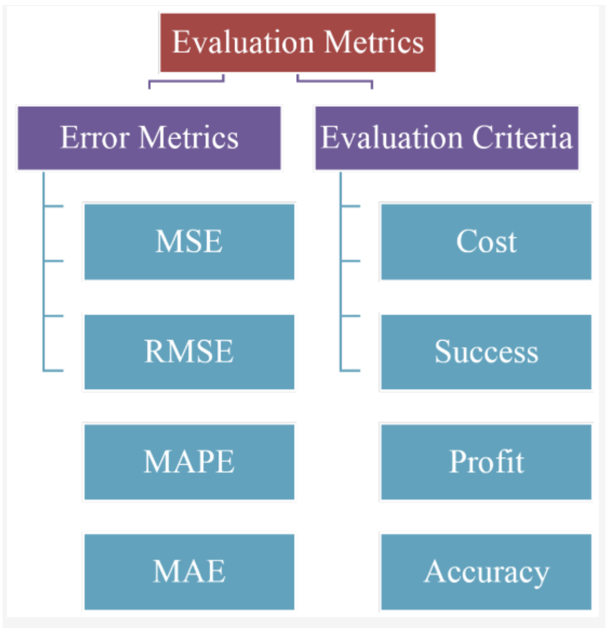


Figure 10: Evaluation Metrics
[3]

4.1.1 ARIMA

Metric Name	MSE	RMSE	MAE	MAPE (%)
unique_collection_ids	39.2838	6.2677	5.0292	26.5276
unique_machine_ids	55.7764	7.4684	5.9804	28.7074
sum_avg_cpu	0.0106	0.1031	0.0769	74.2101
sum_avg_memory	0.0028	0.0527	0.0408	58.6582
sum_max_cpu	0.1664	0.4079	0.3025	71.4796
sum_max_memory	0.0036	0.0601	0.0464	59.4464
non_unique_collection_ids	3.6105	1.9001	1.4816	∞

Table 2: Evaluation Metrics for ARIMA

4.1.2 XGBoost

Metric	MSE	RMSE	MAE	MAPE
Unique Collection IDs	16.7735	4.0955	3.2306	16.84%
Non-Unique Collection IDs	2.0232	1.4224	1.1223	inf
Unique Machine IDs	22.5837	4.7522	3.7426	17.53%
Sum Avg CPU	0.0051	0.0712	0.0540	50.80%
Sum Avg Memory	0.0012	0.0345	0.0262	34.65%
Sum Max CPU	0.0789	0.2809	0.2066	47.11%
Sum Max Memory	0.0015	0.0389	0.0295	34.34%

Table 3: Evaluation Metrics for XGBoost Before Additional Features

Metric	MSE	RMSE	MAE	MAPE
Unique Collection IDs	0.6599	0.8123	0.5722	2.95%
Non-Unique Collection IDs	0.0235	0.1531	0.0486	inf%
Unique Machine IDs	1.0654	1.0322	0.7133	3.31%
Sum Avg CPU	0.0002	0.0133	0.0090	7.81%
Sum Avg Memory	0.0001	0.0076	0.0049	6.36%
Sum Max CPU	0.0035	0.0589	0.0367	7.86%
Sum Max Memory	0.0001	0.0090	0.0061	6.94%

Table 4: Evaluation Metrics for XGBoost After Additional Features

4.1.3 LSTM

Metric	MSE	RMSE	MAE	MAPE
Unique Collection IDs	19.8462	4.4549	3.5425	18.64%
Unique Machine IDs	27.7796	5.2706	4.1924	20.11%
Sum Avg CPU	0.0059	0.0766	0.0591	57.83%
Sum Avg Memory	0.0015	0.0384	0.0296	40.34%
Sum Max CPU	0.0912	0.3019	0.2224	53.01%
Sum Max Memory	0.0018	0.0423	0.0332	41.97%
Non-Unique Collection IDs	1.9758	1.4056	1.1103	inf%

Table 5: Evaluation Metrics for LSTM

When multiple zero values are included in the MAPE calculation for non-unique collection IDs, the outcome is NaN (Not a Number).
The fundamental properties of the data are the cause of the discrepancies in forecast accuracy between collection/machine counts and usage metrics (such CPU and memory):

- **Value Range:** Counts ranging from 1 to 40 provide models with a larger range to identify patterns, resulting in more precise forecasts. Larger errors occur with usage measures, which have much narrower ranges (e.g., sum_avg_cpu: 0-0.8) and disproportionately affect scale-sensitive metrics like MAPE.
- **Aggregation Over Intervals:** When usage indicators are aggregated over 300-second intervals, the resulting smaller fluctuations and smoother patterns make it more difficult for models to identify sudden, abrupt spikes. Counts, on the other hand, show more distinct patterns and are therefore easier to forecast.

Out of the three studied algorithms (XGBoost, LSTM, and ARIMA), XGBoost performs the best in terms of resource utilization metrics predictions. When further features are added, including lag values, rolling statistics, and time-based features, the improved performance becomes even more apparent. These characteristics give XGBoost historical context and patterns, which improves learning efficiency and prediction accuracy. By utilizing the available data and its boosting process, which iteratively improves predictions by learning from the errors of previous iterations, XGBoost works well even without these extra features. All the metrics demonstrate how much improved accuracy XGBoost obtains with the addition of these enhanced features, demonstrating the robustness of the algorithm in managing complex patterns in data.

Justification for larger errors in ARIMA compared to LSTM
Metrics Sensitivity: The smooth predictions generated by LSTM often lead to fewer large errors, improving MSE, RMSE, and MAE scores. However, these metrics do not account for LSTM's inability to accurately predict sudden surges.
ARIMA's Time Shift: The small time shift in ARIMA forecasts causes spikes to be recorded but not aligned with the actual data, increasing error metrics. While spike detection may visually appear better, this misalignment results in worse metrics.

4.2 Data Visualisation and Cloud Insights

Given that the dataset spans a total duration represented by 1e6 seconds, the number of intervals created is determined by dividing the total time by the interval length:
Number of Intervals = 1e6/300 = 333.33
This computation yields about 333 intervals, which is marginally less than the 501 samples that are utilized in sample index charting. The reason for the disparity is that in real-time interval analysis, data is aggregated within each 300-second window, whereas in sample indices, each data point is considered as a separate unit of analysis.
Libraries used in Data Visualisation were :
Matplotlib: A complete Python library for making static, animated, and interactive visualizations is called Matplotlib. Plots, charts, histograms, and other visual data representations are frequently created with it.
Seaborn: Seaborn is a data visualization package that offers an advanced interface for producing eye-catching and educational statistical charts. It is developed on top of Matplotlib. With its pre-built themes, color schemes, and capabilities for displaying data relationships, distributions, and categorical variables, Seaborn makes the process of producing complex plots easier. With just a few lines of code, users may create visual representations of data in a pandas DataFrame, making it especially well-suited for this type of work.

4.2.1 ARIMA

To start with, the ARIMA (AutoRegressive Integrated Moving Average) model was used to predict all the metrics obtained from time-series data concerning the utilization of system resources. Using past data as a basis, the ARIMA model was tweaked with the parameters (5,2,0)—where 5 represents the number of lag observations (p), 2 represents the degree of differencing (d), and 0 represents the moving average window size (q)—in order to forecast future values. :

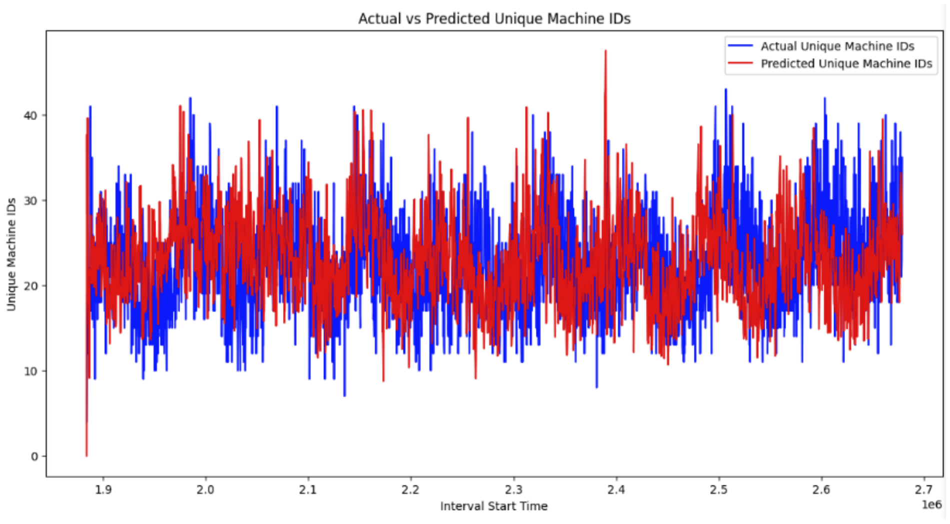


Figure 11: ARIMA Unique Machine Ids

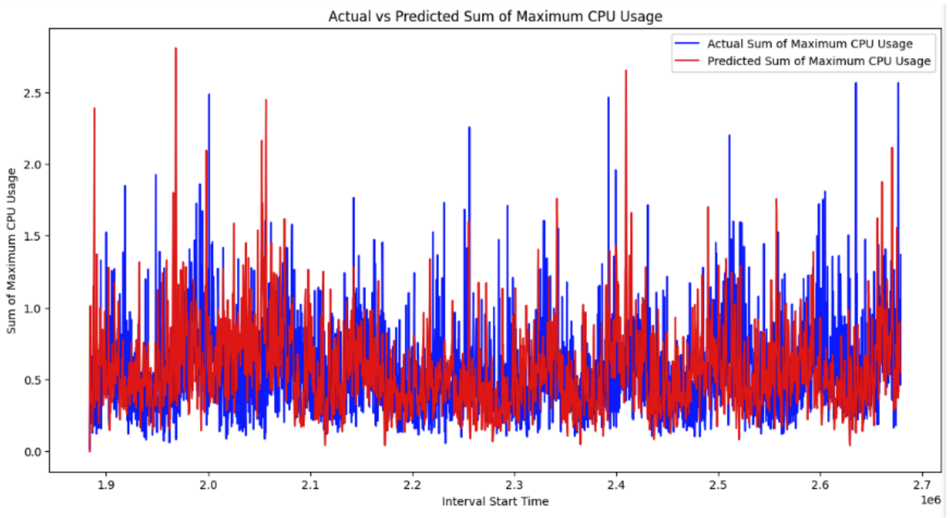


Figure 12: ARIMA sum of max cpu

Every metric in the dataset was projected using the appropriate ARIMA model, and the outcomes were contrasted with the real values. Plots for unique machine ids, sum max CPU are included in this evaluation as an example in Figure 11 and 12 to show the effectiveness of the ARIMA model. Different hyperparametertuning comparisons will also be used for further analysis in the ARIMA prediction. Cloud Insights are demonstrated based on these visualisations in the next part.

Cloud Insights

The unique machine ID visual representation exhibits a highly irregular pattern with regular ups and downs. This implies that the quantity of machines that are actively involved in the cloud environment varies quickly over time. The overall trend of the actual values (blue line) is often followed by the anticipated values (red line). It is usual for time series forecasting to exhibit small lags or lead at specific points, though, in the projections.
The number of unique machine IDs in use periodically rises, suggesting spurts of cloud activity. This indicates that extra machines are being deployed or activated in response to specific events, such as processing tasks that are in high demand or scheduled jobs.An indication of the model's accuracy is the close with which the predicted values match the actual values. Nonetheless, there is an obvious variation in timing between the observed and projected increases. This implies that even if the model performs well in estimating the total number of machines, it might not be as accurate in foretelling the precise moment of these spikes.

Similar to the first unique machines, there are periodic spikes in CPU usage that correspond with the spikes in the number of machine IDs. This suggests that the CPU use naturally rises with the number of machines deployed (maybe as a result of increased demand. Though they suffer from the same time shift as machine IDs, the forecasts for CPU utilization likewise fit actual numbers quite well. This supports the notion that, although the model does a good job of predicting the size, further fine-tuning of the time is required.

The quantity of distinct machine IDs and CPU use appear to be strongly correlated. As expected in a cloud setting where resources scale with demand, a bigger number of machines usually translates into increased CPU consumption for any given interval.The machine ID and CPU consumption prediction models are operating similarly, but with a slight delay in forecasting the spikes. This consistency shows that both forecasts may be influenced by the same underlying factors or data, and that refining one model (by lowering the temporal shift, for example) may help improve the other.

During a normal window of time (let's say from 2.0e6 to 2.1e6), you may notice that the mean number of distinct machine IDs varies by about 20 to 25 computers.The total maximum CPU consumption also tends to settle during these periods, ideally in the range of 0.5 to 1.0 (arbitrary units based on the graph's size).This indicates that the associated CPU consumption stays pretty stable within this range when you have an average of 20–25 active computers. This can be used to forecast future resource needs for workloads that are comparable.

When there is a spike in the number of unique machine IDs, such as from time 2.2e6 to 2.3e6, at which point the machine count may reach a maximum of 40 machines.The CPU utilization also exhibits a notable peak in response to this surge, possibly reaching values of 1.5 to 2.0.The CPU use increases noticeably at the height of the machine count. This implies that high-intensity processing jobs or a spike in demand are indicated by a direct correlation between spikes in CPU demand and spikes in machine count.The number of machines may decrease to 10–15 during periods when there are few unique machine IDs, such as 2.4e6.The CPU utilization also decreases dramatically during these times of low activity, maybe to levels between 0.2 and 0.5.This could be an indication of off-peak hours or periods when fewer tasks are in progress, which could result in resource reductions and cost savings.
This helps in understanding the cloud environment's behavior under different workloads, allowing for better resource management.

When combined with machine IDs, the patterns in unique and non-unique collection IDs offer important insights into the dynamics of workload in a cloud context. Better forecasting and management of cloud resources are made possible by the significant correlation between both measures, which shows that an increase in one usually corresponds with an increase in the other. In order to achieve optimal performance and cost effectiveness, non-unique collection IDs draw attention to the necessity of parallel processing, which can inform decisions about resource allocation and task scheduling.

Additional Features

In an effort, to increase the accuracy and robustness of our prediction modeling, we improved the dataset by include **rolling statistics, lag features, and time-based features.**[46][51][49] The purpose of these improvements was to improve the ARIMA model's ability to represent temporal patterns and relationships in the data.

The model's performance was not considerably enhanced by the addition of these elements, which proves that ARIMA already includes techniques to account for temporal dynamics, such as moving averages and autoregression. This result shows that certain patterns can be handled by ARIMA's built-in capabilities, which do not require extensive feature engineering.

4.2.2 Xgboost

The XGBoost technique similar to ARIMA to predict all metrics. As an example we visualize a couple here : sum max memory usage and Unique Collection IDs in Figure 13 and 14

Because it can effectively simulate non-linear interactions, XGBoost is an ensemble machine learning method based on decision trees that performs exceptionally well at predicting complicated patterns.To provide predictions, it makes use of a variety of attributes that are extracted from the data.
The performance of the model was significantly improved by the addition of lag, time-based, and rolling statistics elements; the consumption metrics showed the greatest improvement.

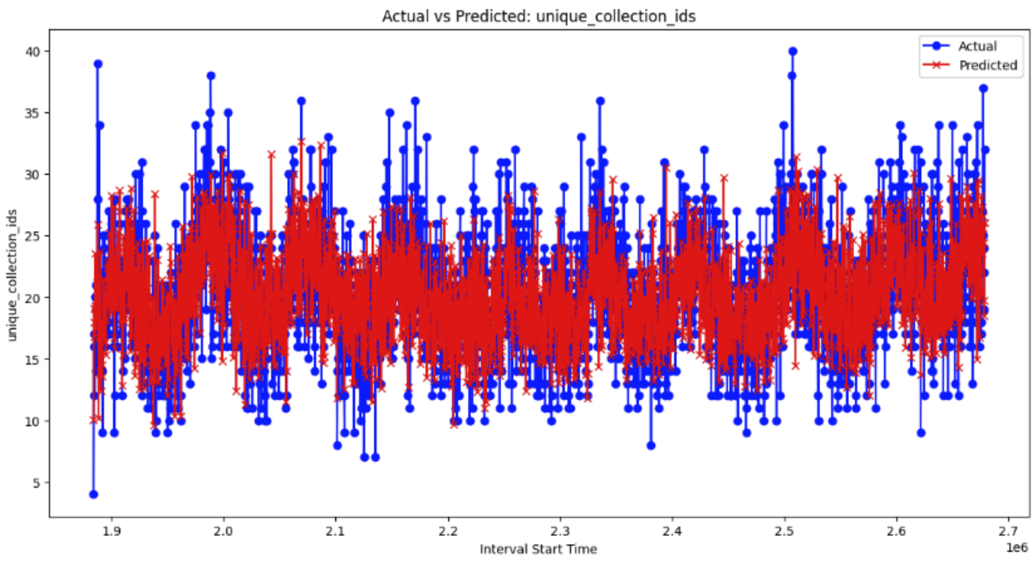


Figure 13: Xgboost Unique Collection ids

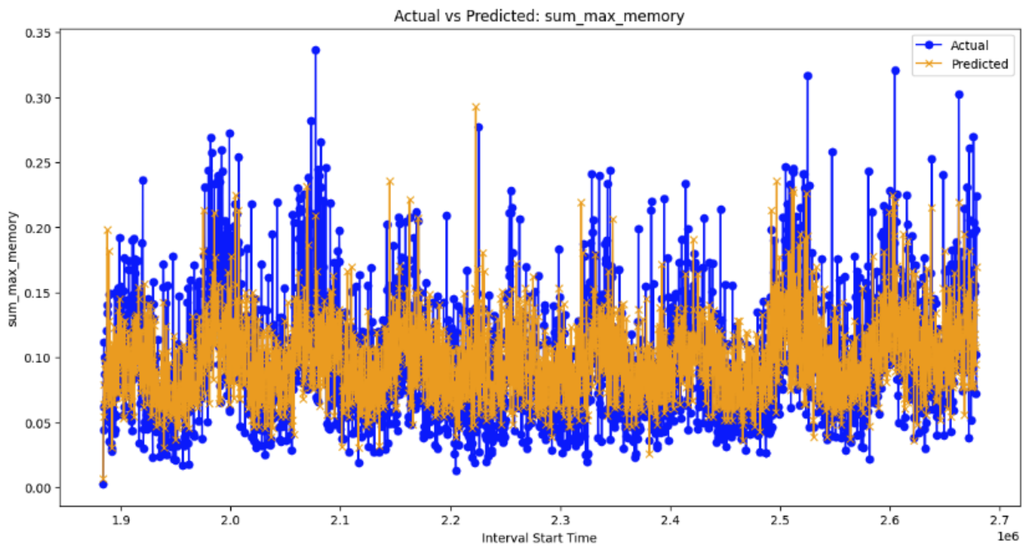


Figure 14: xgboost sum max memory

Additonal Feature Engineering

When lag features, rolling statistics, and time-based features are added to the model, XGBoost—which is well-known for its efficiency in managing intricate feature interactions and non-linear relationships—performs noticeably better. In comparison, models such as LSTM and ARIMA do not show the same level of performance improvement with the addition of similar characteristics as described in their sections respectively.

Lag Features: To better capture temporal relationships and trends that may affect future values, the model incorporates lagged versions of key variables (such as CPU and memory consumption, unique and non-unique collection IDs, and CPU and memory usage). This works especially well with XGBoost, whose ability to anticipate future results depends on the explicit input of past values.[46]
Rolling Statistics: To give the model information on the short-term moving averages and variability, rolling means and standard deviations of important variables are computed. In time series forecasting, local trends and volatility are crucial, and this helps XGBoost take them into consideration.[51]
Time-Based Features: To help the model identify daily and weekly patterns in the data, features like the day of the week and hour of the day are incorporated.[49]

Data Visualization for the same features i.e. Unique Collection ids and Sum of maximum memory is shown in Figure 15 and Figure 16. We can clearly see the enhancement in predictions visually after adding these additional features which provide extra context about the underlying patterns in the data.

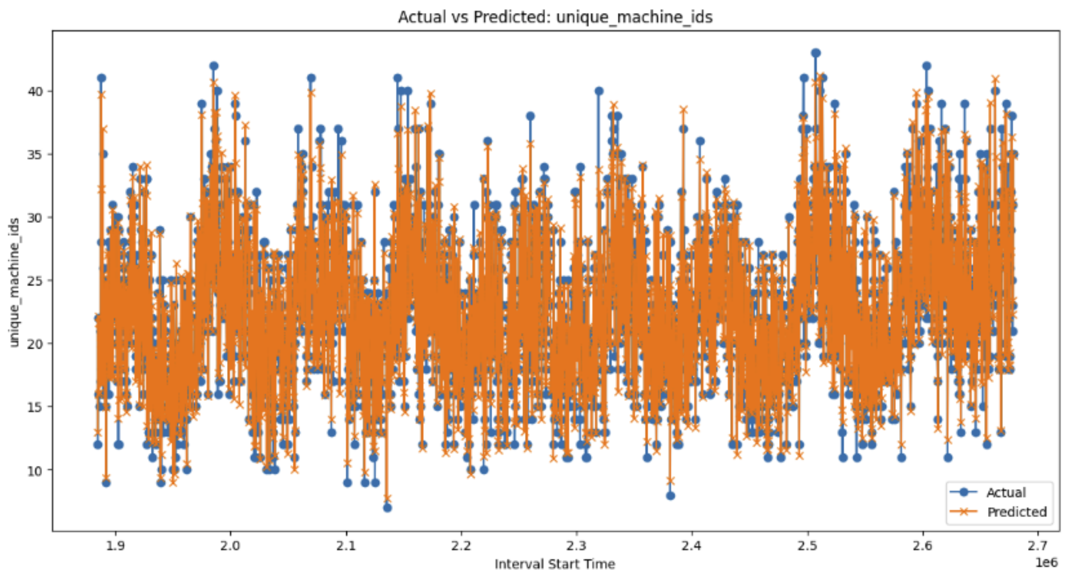


Figure 15: xgboostadditonalfeatures unique machine ids

Count-based metrics have a tendency to be less smooth and more distinct. Examples of these metrics are the number of unique machines, non-unique collection IDs, and unique collection IDs. These counts, which are frequently integer-based, show simpler patterns. As a result, these count metrics were quite well estimated before to the addition of the additional features, and they continued to be more accurately modeled thereafter.

In contrast, the accuracy of the model was significantly enhanced by the addition of the additional features, especially for the continuous consumption metrics (sum max cpu, sum max memory, sum avg cpu, and sum avg memory). These measurements have substantially lower initial accuracy, ranging from roughly 50 percent to 66 percent But the accuracy increased to 92 to 94 percent after adding latency, time-based, and rolling statistics features.This is because , Continuous data exhibits smooth changes and trends that are more difficult to model, such as CPU and memory utilization.With these sophisticated features, the model was able to more accurately represent historical patterns and trends and comprehend the smooth variations within the restricted range of [0, 1]. This improvement results in significantly higher prediction accuracy and represents a deeper comprehension of the temporal dynamics of the continuous data.

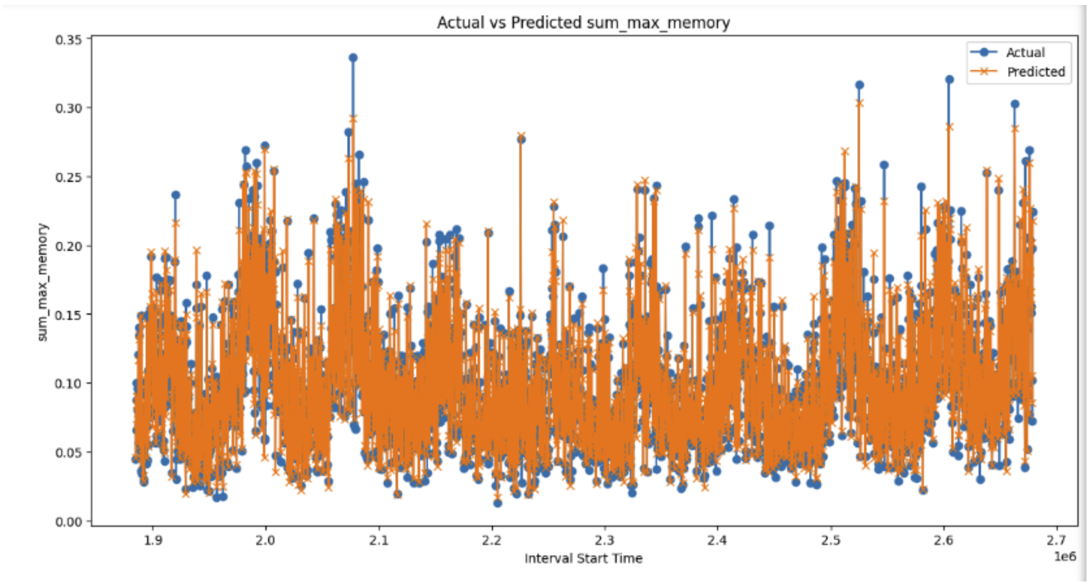


Figure 16: xgboostadditonalfeatures sum max memory

Cloud Insights

Cloud Insights are demonstrated based on Figure16 and 15 i.e. after applying the additional features. Memory utilization clearly exhibits periodic spikes, with many peaks that seem to happen at rather regular intervals. These peaks can be associated with times of heavy load or certain tasks that use additional RAM. Similar to this, there are recurring surges in the quantity of unique collection IDs. Collection IDs have a strong correlation with machines, meaning that there is a considerable increase in activity (i.e., more machines being used) at specific times.

There appears to be a correlation between memory usage spikes and unique collections, indicating that memory usage peaks during periods of high activity (more collections/machines). It makes sense because more active computers would need more RAM. For example, you will also notice a high in memory utilization if you see a spike in the number of unique collections at interval 2.2M.Significant peaks can be seen in both collections and memory at specific intervals, such as 2.0M, 2.2M, and 2.5M. These intervals may indicate periods of high traffic, maintenance windows, batch processing, or other reasons that put a lot of strain on the cloud infrastructure. These need to be closely watched for any opportunities for optimization or to proactively commit more resources.

Memory utilization appears to be consistent within the range of 0.1 to 0.15 when the number of collection IDs (representing machines) is approximately 20–25. But memory use increases dramatically, frequently above 0.2, if the number of collection IDs approaches 30 or more.There appears to be a comparable increase in memory consumption of about 0.05 to 0.1 for every additional 10-15 collection IDs. Based on the number of machines that are currently in operation, this can be used as a ballpark estimate to predict memory utilization.Plan for memory use to potentially surpass 0.2 during intervals where collection IDs reach 30, which may require scaling up cloud resources during these times.

4.2.3 LSTM

Initially, like ARIMA and XGBoost, the Long Short-Term Memory (LSTM) model was used to forecast several metrics generated from time-series data linked to cloud resource utilization. Recurrent neural networks (RNNs), of which LSTMs are one kind, are made to interpret sequential input by identifying complex patterns across time and storing long-term dependencies. We provide samples of unique collection ids and unique sum of average CPU time for the visualisation as shown in Figure 17 and Figure 18.The power of LSTM resides in its capacity to predict and analyze underlying trends and seasonality in data with great accuracy, offering insightful information for proactive resource management and planning in cloud systems.

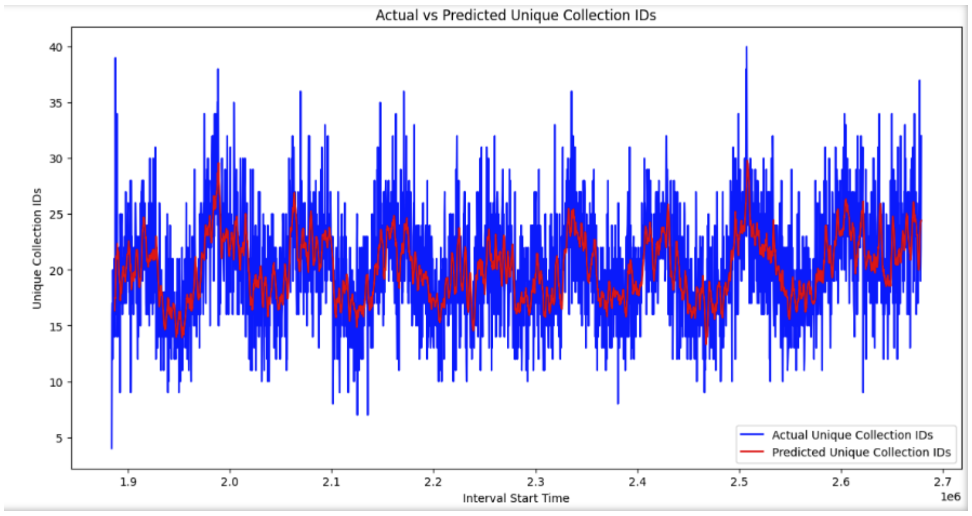


Figure 17: LSTM unique Collection IDs

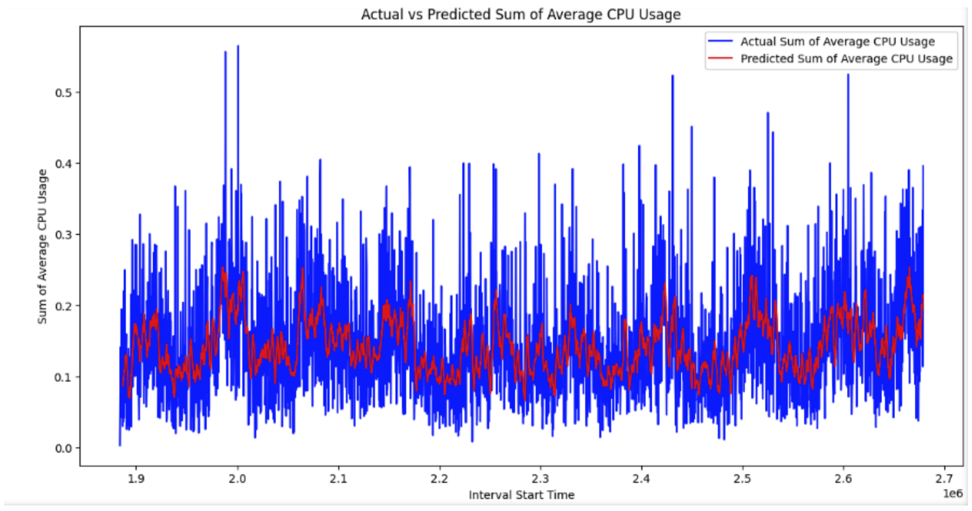


Figure 18: LSTM sum of avg cpu

The LSTM model is less sensitive to sharp spikes or anomalies in the data, though, as it has a tendency to smooth out abrupt variations. In the Sum of Average CPU Usage graph, the LSTM predictions (represented by the red line) show a more leveled response, frequently underestimating the intensity and timing of these spikes, whereas the actual data (represented by the blue line) exhibits sharp and irregular spikes, possibly caused by unforeseen high-load events or sudden surges in user activity. The smoothing effect may result in under-provisioning during periods of peak demand, which could affect performance and create difficulties for real-time resource allocation.

Despite this drawback, the LSTM model's ability to enable long-term forecasting and highlight persistent usage patterns makes it extremely valuable for strategic planning. For instance, the model accurately predicts rising patterns in CPU utilization and Unique Collection IDs, indicating times when increased resource allocation might be advantageous to sustain peak performance. Since a rise in Unique Collection IDs frequently corresponds with higher CPU utilization, the LSTM forecasts can help foresee growth phases and make judgments about preemptively allocating additional compute resources to ensure smooth scalability and enhanced user experience.

Cloud Insights

Upon examining the figures, it is evident that the LSTM model is highly capable of capturing general trends in CPU utilization and the quantity of Unique Collection IDs. For example, the model is able to effectively represent the steady increases and dips that correspond to typical workload patterns throughout the intervals between 2.0M and 2.5M, demonstrating its ability to accurately depict consistent growth or drop over long periods of time. Furthermore, the model is also good at detecting seasonal trends, including frequent batch processing operations, weekly maintenance chores, or recurrent peaks and troughs that happen at regular intervals. These insights are essential for predicting regular resource requirements and making sure the cloud infrastructure is configured appropriately to support workloads that are anticipated.

4.3 Residuals

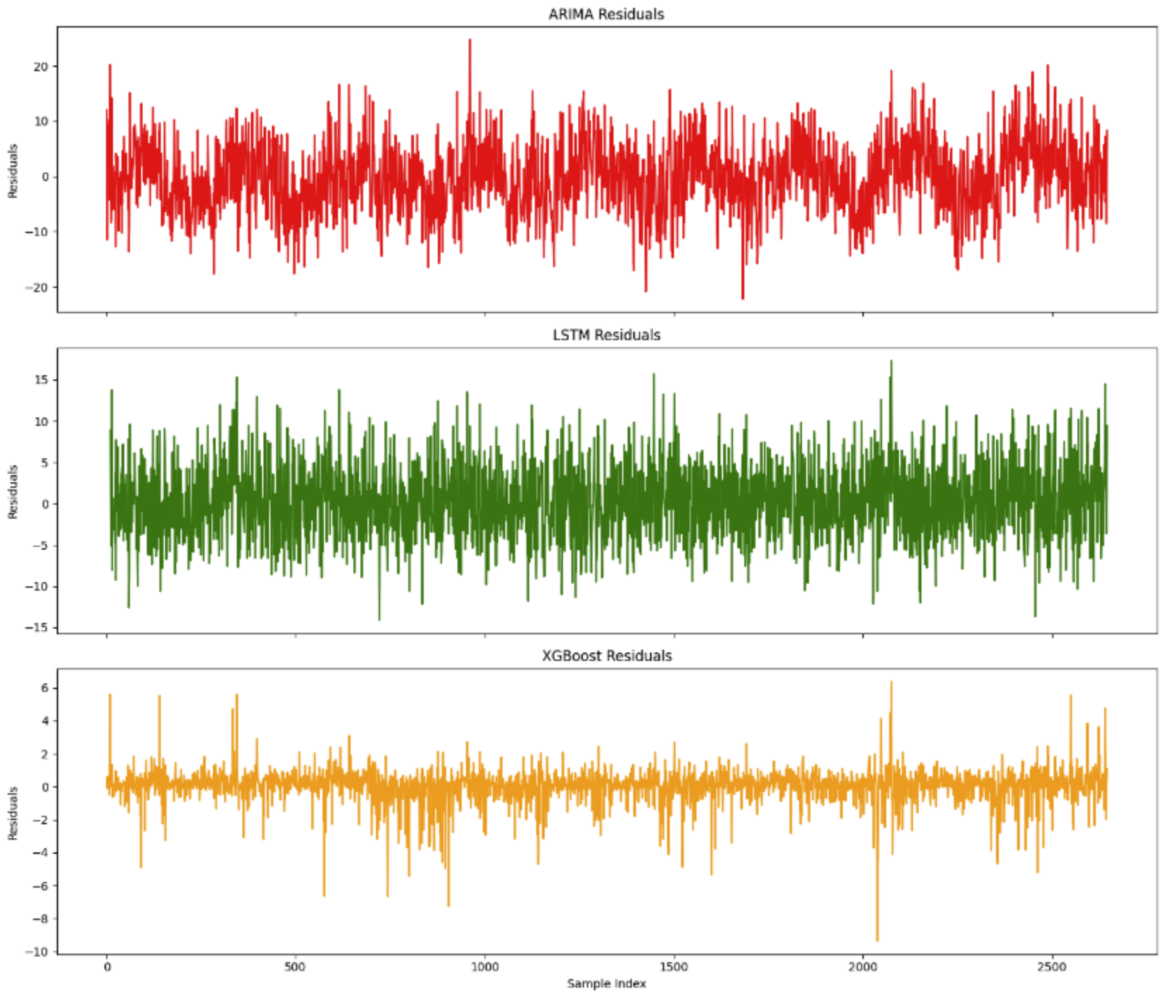


Figure 19: Residuals

The discrepancy between a model's predicted and actual values is represented by residuals. By showing where and how much the predictions differ from reality, they offer vital insight into how well a forecasting algorithm is performing. Residuals are significant because they show the model's capacity to catch abrupt changes or spikes in the data and assist in identifying patterns in prediction mistakes, such as persistent over- or underestimation.[41][37]

Residuals provide a clear view of the advantages and disadvantages of each model when comparing ARIMA, LSTM, and XGBoost for cloud resource management. Though timing shifts cause bigger residuals and decreased accuracy, ARIMA is especially good at recognizing unexpected spikes, which is crucial in dynamic cloud environments. While LSTM is good at identifying trends and seasonality, it may miss sudden spikes since it generates smoother predictions with fewer significant errors. When it comes to real-time resource management, when accuracy and responsiveness are critical, XGBoost is the most dependable option because it consistently maintains low residuals and predicts spikes properly without experiencing major timing problems.

4.4 Hyperparameter Tuning

Time series forecasting relies significantly on hyperparameter tweaking because the precise setting of forecasting models' hyperparameters greatly affects their performance. To properly capture the patterns, trends, and seasonality in time series data, these models—each with its own set of parameters—need to be fine-tuned.Appropriate parameter selection is crucial for effectively simulating the link between present and previous values in models with autoregressive components. Poor forecasts might result from incorrect parameter settings, particularly when there are intricate temporal connections.[7]

Finding the combination of hyperparameters that produces the highest model performance is the aim. We used a random search strategy to find the best hyperparameters for the LSTM, XGBoost, and ARIMA algorithms. Rather of analyzing every potential configuration, random search is an effective technique that samples random combinations of parameters to explore the hyperparameter space.[35]

4.4.1 ARIMA

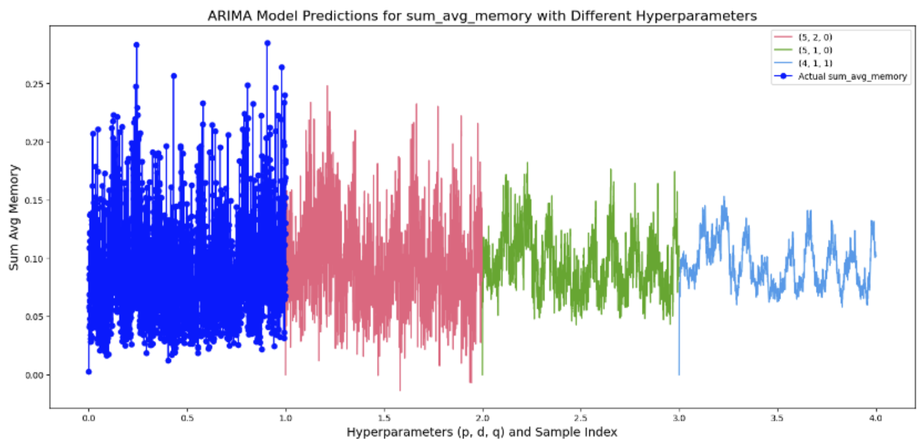


Figure 20: ARIMA Hyperparameter Tuning

The goal of hyperparameter tuning for ARIMA (AutoRegressive Integrated Moving Average) models is to determine the best values for the (p, d, and q) parameters. The model's complexity and capacity to identify patterns in time series data are determined by these parameters. For hyperparameter tuning of all the features, the following three sets of hyperparameters were hardcoded after performing a few random searches to visualize the results in Figure 20:

- **(5,2,0):** The model considers the latest five observations ($p = 5$), differences the data twice to achieve stationarity ($d = 2$), and has no moving average component ($q = 0$). This model may introduce considerable errors owing to overfitting, but it was chosen since it is good at capturing the overall trend and amplitude of peaks. The model's propensity to time-shift these peaks suggests that, although it does a good job of capturing the magnitude, it has trouble with exact timing.

- **(5,1,0):** Similar to the first set, but with the data differenced once ($d = 1$).A smoother prediction curve is produced by the model with first-order differencing (d=1), which lessens some of the overfitting observed in the (5, 2, 0) model. Still, it fails to capture the exact peaks and valleys that are essential to our research.

- **(4,1,1):** The model considers the latest four observations ($p = 4$), differences the data once ($d = 1$), and includes a moving average component with a window size of one ($q = 1$). A moving average component (q=1) in this model smoothes out the prediction and reduces noise in the trends it catches. When it comes to peak detection, it might not perform as well as the (5, 2, 0) model. Although this model minimizes the time shift problem observed in the (5, 2, 0) model and provides the greatest fit for predicting overall trends, it may not be as good at capturing the abrupt spikes.

With p=5, d=2, and q=0, the combination performed the best. The model's accuracy increased as a result of this combination's successful capture of the important patterns and trends in the time series data.

4.4.2 Xgboost

Hyperparameters in Xgboost used are :
n estimators: The number of boosting rounds or trees to build.
maximum depth: A control on the complexity of the model that indicates the maximum depth of each tree.
Learning Rate: The weights of the model are learned using this step size.

The following sets of parameters were evaluated and demonstrated in Figure 21:

```
Set 1: {'objective': 'reg:squarederror', 'eval_metric': 'rmse', '
    n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1}
Set 2: {'objective': 'reg:squarederror', 'eval_metric': 'rmse', '
    n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.01}
Set 3: {'objective': 'reg:squarederror', 'eval_metric': 'rmse', '
    n_estimators': 150, 'max_depth': 7, 'learning_rate': 0.05}
```

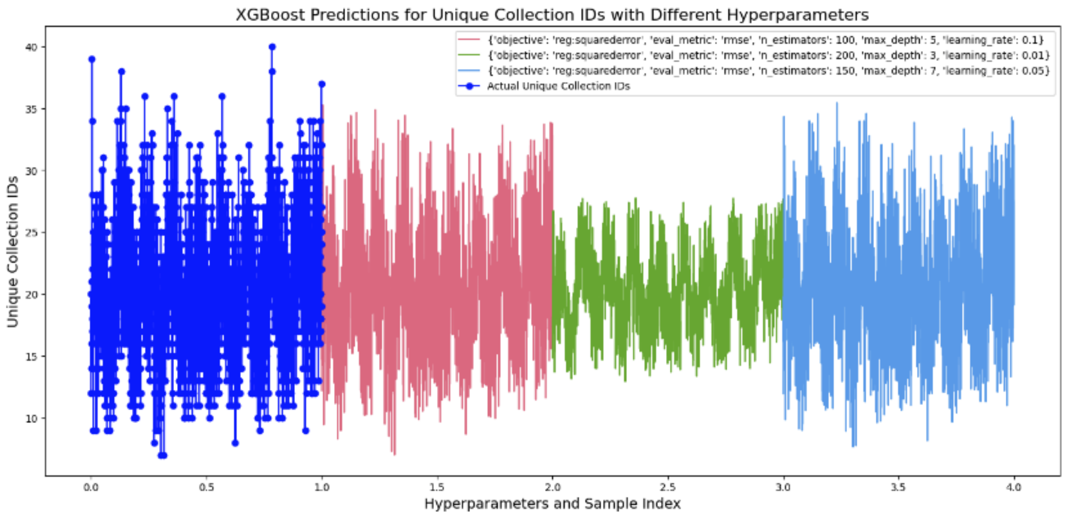


Figure 21: Xgboost Hyperparameter Tuning

- **n_estimators=100, max_depth=5, learning_rate=0.1:** While this configuration smooths out a lot of the variability observed in the actual data, this arrangement tends to produce modest variations, catching some spikes. A good number of estimators (`n_estimators=100`) and a moderate depth of the trees (`max_depth=5`) strike a balance between preventing overfitting and catching intricate patterns.

- **n_estimators=200, max_depth=3, learning_rate=0.01:** This configuration produces a prediction line that is smoother and less reactive since it has a higher number of estimators and a lower max depth. Underfitting is likely to occur because this set of hyperparameters is too conservative for this specific time series.

- **n_estimators=150, max_depth=7, learning_rate=0.05:** A more aggressive model with deeper trees and a modest learning rate is provided by this option. Compared to the other two designs, it captures spikes more successfully and closely resembles the real spikes, reacting more strongly to changes in the data.

For catching the spikes and sudden changes which is essential in a cloud DC, the third set (n estimators=150, max depth=7, learning rate=0.05) works best. This configuration is in line with your requirements for managing resource allocation in the cloud environment and detecting sudden changes.

4.4.3 LSTM

Hyperparameter tuning for LSTM (Long Short-Term Memory) involves:
units: Each LSTM layer's total number of units (neurons), which affects the model's ability to learn from data.
layers: The quantity of LSTM layers arranged in a stack, which influences the model's intricacy and depth.
epochs: The total number of iterations over the whole dataset, or training epochs.
batch_size: The quantity of samples handled before an update to the internal parameters of the model.

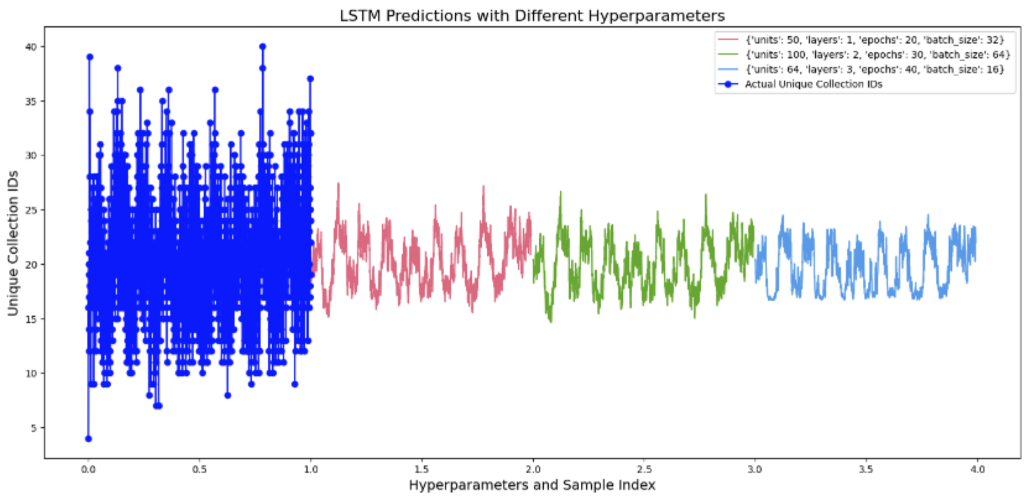


Figure 22: LSTM Hyperparameter Tuning

The following hyperparameter sets were evaluated and demonstrated in Figure 22:

```
Set 1: {units: 50, layers: 1, epochs: 20, batch_size: 32}
Set 2: {units: 100, layers: 2, epochs: 30, batch_size: 64}
Set 3: {units: 64, layers: 3, epochs: 40, batch_size: 16}
```

- **Set 1:** `units = 50, layers = 1, epochs = 20, batch_size = 32`
  - Due to the single layer and fewer units, this architecture is probably good at capturing broad patterns but may have trouble handling longer-term or more complicated dependencies.
- **Set 2:** `units = 100, layers = 2, epochs = 30, batch_size = 64`
  - Though it may still miss abrupt changes, this setup is predicted to capture patterns and seasonality more effectively than the simpler model.
- **Set 3:** `units = 64, layers = 3, epochs = 40, batch_size = 16`
  - Because of the fundamental characteristics of LSTMs, this arrangement should be good at catching both long-term and short-term patterns, however it may still have trouble with sudden shifts.

4.5 Correlation Matrix

A dataset's correlation matrix can be used to determine the relationships between its various variables. For example, it can show the potential relationships between different elements such as CPU, memory, or the number of unique machine IDs in the context of time series forecasting. When two variables have a strong correlation, it could be possible to predict one from the other or for there to be an underlying element influencing both of them.

In the area of resource management, particularly for systems such as cloud computing environments where memory, CPU, and other resources are handled, it is imperative to make well-informed judgments regarding scaling, or the distribution of resources either way. To optimize these choices, a correlation matrix generated from projected values can be quite helpful.As an example table 6 shows Correlation Matrix of Predicted values using Arima. This can be referred in the scaling decisons section for Table 8

	p_uic	**p_umid**	**p_sac**	**p_sam**	**p_smc**	**p_smm**	**p_nuic**
p_uic	1.0000	0.9499	0.4661	0.6010	0.4912	0.6312	0.3413
p_umid	0.9499	1.0000	0.4585	0.6179	0.4821	0.6457	0.5885
p_sac	0.4661	0.4585	1.0000	0.7067	0.8501	0.7148	0.1970
p_sam	0.6010	0.6179	0.7067	1.0000	0.7068	0.9766	0.3359
p_smc	0.4912	0.4821	0.8501	0.7068	1.0000	0.7298	0.2061
p_smm	0.6312	0.6457	0.7148	0.9766	0.7298	1.0000	0.3455
p_nuic	0.3413	0.5885	0.1970	0.3359	0.2061	0.3455	1.0000

Table 6: Correlation Matrix for Predicted Values of ARIMA

For scaling decisions, the following information is essential:
Scale Up: You may choose to allocate more memory ahead of time in preparation of the increased load if your model indicates a notable rise in CPU utilization, and this is highly connected with memory usage.
Scale Down: On the other hand, if a decrease in one resource's use is anticipated and it is associated with another, you may want to think about reducing the use of both resources in order to save money. This strategy can be applied in an automated scaling system, which makes decisions in real time without the need for human interaction based on thresholds and correlations.

4.6 Key Differences in Model Evaluation

Figure 23, Figure 24 and Table 7 depicts the Data Visualisation and differences in evaluation respectively for all of the 3 modelling Techniques used in this study.

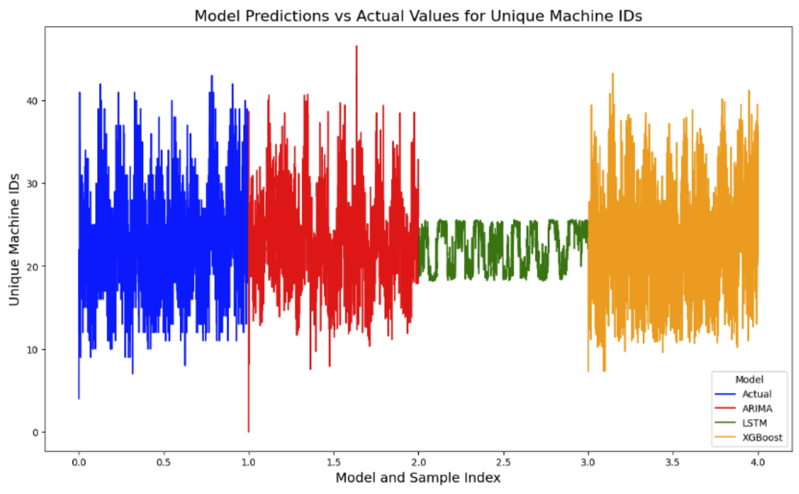


Figure 23: All model prediction for Unique Machine Id

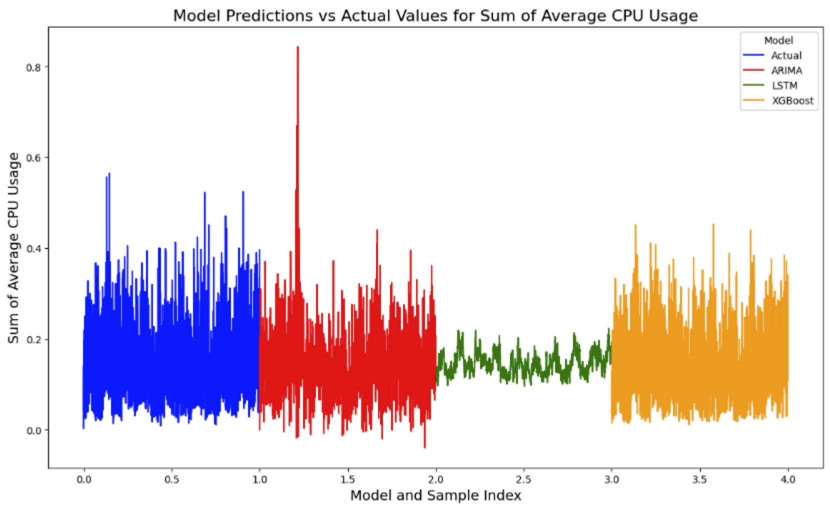


Figure 24: All model prediction for Sum of Average CPU

Aspect	**ARIMA**	**XGBoost**	**LSTM**
Model Characteristics	Captures linear trends and periodicity; struggles with non-linear patterns	Excels at capturing complex, non-linear relationships; well-suited for diverse metrics; effectively captures trend and seasonality	Captures long-term dependencies; smooths out sharp changes; effectively captures trends and seasonality
Prediction Accuracy and Spikes	Detects spikes with a slight time shift; may misalign with actual data, particularly during sudden changes	Accurately predicts sharp spikes; adapts well to abrupt changes; most reliable for visualizing and forecasting spikes	Smooths predictions; may miss sharp spikes, but closely follows the overall trend and periodicity of the actual data
Application in Cloud Resource Management	Suitable for long-term planning; needs adjustments for spike timing	Effective for real-time management; crucial for handling sudden demand spikes	Useful for identifying general patterns and seasonal components; less reliable for immediate decisions during volatility
Impact of Feature Engineering	Minimal benefit from external features; focuses on inherent patterns	Significant improvement with feature engineering; effective cross-target prediction	Limited benefit from external features; internal memory captures patterns; good at identifying seasonality within the data
Error Metrics	Larger errors due to time shifts in spike predictions, leading to increased MSE, RMSE, and MAE	Lowest error metrics (MSE, RMSE, MAE) due to effective handling of complex patterns; further improved with additional features	Moderate error metrics; fewer large errors due to smoothed predictions, but struggles with sudden surges, impacting accuracy in volatile conditions
Trend and Seasonality Handling	Captures the general trend but often shows timing shifts; struggles with seasonality, leading to less accurate cyclical predictions	Handles complex trends and seasonality effectively without significant timing issues; accurately captures both trends and seasonality	Closely follows the general trend of the data; captures seasonality effectively, aligning closely with cyclical patterns, though it smooths out some fluctuations

Table 7: Comparison of ARIMA, XGBoost, and LSTM in the Context of Cloud Resource Utilization Prediction

4.7 Scaling Decisons

Predictive analytics and machine learning models can be used to forecast resource utilization and guide scaling decisions in order to meet these needs.
Below is an example in Table 8 of first few intervals for the test data where scaling is necessary based on Arima predictions -

Index	Scaling Action	Metric
0	Scale Down	non_unique_collection_ids
1	Scale Up	unique_machine_ids
2	Scale Down	non_unique_collection_ids
3	Scale Up	non_unique_collection_ids
4	Scale Down	unique_machine_ids
5	Scale Down	sum_max_cpu
6	Scale Down	non_unique_collection_ids
7	Scale Down	sum_max_cpu
8	NaN	NaN
9	Scale Up	sum_avg_cpu
10	NaN	NaN
11	Scale Up	sum_avg_memory
12	Scale Up	sum_max_cpu
13	Scale Down	non_unique_collection_ids
14	Scale Down	non_unique_collection_ids
15	Scale Down	non_unique_collection_ids

Table 8: Scaling Actions and Corresponding Metrics

The thresholds have been set to initiate scaling measures upon exceeding or lowering of predetermined values in projected consumption.

Scale Up Threshold: A predetermined "scale-up" threshold is exceeded by the forecasted value of a measure, indicating a considerable rise in demand. In these situations, more resources ought to be allocated to manage the heightened demand.
Scale Down Threshold: On the other hand, if the anticipated value is less than a "scale-down" threshold, it implies that a decline in demand is anticipated. This offers a chance to reallocate resources, which lowers expenses without sacrificing the caliber of the services.

A more sophisticated method is examining the correlations between the anticipated values of several measures, going beyond basic threshold-based choices. Using the predicted numbers, a correlation matrix may be formed to help determine the connections between various resource demands.

As an example we can refer to table 6 :-
Positive Correlation: When there is a significant positive correlation between two metrics, an increase in one may be followed by an increase in the other.
Negative Correlation: When there is a significant negative correlation between two measurements, it may be possible to link a rise in one to a fall in the other.

As we can observe in the correlation matrix in Table 6:
There is a high positive connection of 0.7067 between psam (Sum of Average Memory Usage) and psac (Sum of Average CPU Usage). This suggests that higher average CPU usage is frequently linked to higher memory usage.
Metrics with low or negative correlations, on the other hand, (psmc and pnuic, for example, have a correlation of 0.2061) would imply that decisions about scaling for one metric might not necessitate proportionate changes in the other, enabling more focused and effective resource scaling.

5 Conclusion and Future Work

A sophisticated method of managing workload in cloud computing settings through time series forecasting models has been developed in this work. To maximize resource allocation, the study aimed to precisely forecast key workload parameters, such as CPU and memory consumption, as well as the number of jobs/collections and machines, using machine learning algorithms including XGBoost, ARIMA, and LSTM. XGBoost is the most efficient method for workload pattern forecasting, as the examination shows that it performs better in terms of accuracy and reliability than both ARIMA and LSTM. Robust model robustness in managing the dynamic and unpredictable nature of cloud workloads is the result of thorough data analysis, visualization, and hyperparameter adjustment.

This solution's ability to give cloud service providers precise forecasts so they can decide whether to scale pro-actively is a clear indication of its worth. By preventing overprovisioning of resources, this capability not only maximizes resource use but also dramatically lowers operating costs and energy consumption in data centers. The research also highlights how critical it is to manage unexpected surges and data overloads, which are frequent in actual cloud setups. The suggested method helps preserve service performance and dependability by precisely predicting these unexpected changes in demand, guaranteeing that cloud services can quickly and effectively adjust to changing workloads. This flexibility is essential for improving user satisfaction and enhancing cloud operations' sustainability, which in turn helps to ensure the effective and environmentally responsible administration of cloud resources.

To improve workload management and forecasting even further, future work on this project may investigate the integration of edge-to-cloud services and federated learning. Workload prediction models could benefit from federated learning, which may be used to increase the scalability and privacy of the models by training machine learning models across a number of decentralized devices or servers without requiring the transfer of raw data to a central location. Cloud providers might eliminate the requirement for massive data transport to centralized cloud servers by implementing federated learning, which would enable local data processing at the edge. This method would improve forecasting model speed and efficiency in addition to data privacy and security. It would also enable real-time adjustments to task management methods based on localized data insights.[21][22] Furthermore , we can concentrate on merging edge-to-cloud services in addition to federated learning to provide a framework for workload management that is more fluid and flexible. By processing data closer to the source, edge-to-cloud services lower latency and speed up reaction times. The system may handle abrupt spikes or data overloads more quickly and reliably by merging edge computing with cloud-based forecasting algorithms. This would ensure continuous service quality and dependability.[40]

References

[1] Augmented dickey-fuller test usage. `https://rtmath.net/assets/docs/finmath/html/93a7b7b9-e3c3-4f19-8a57-49c3938d607d.htm#ADFUsageSection3`. Accessed: 2024-07-22.

[2] Google bigquery. `https://cloud.google.com/bigquery`. Accessed: 2024-08-26.

[3] Z. Ahamed, M. Khemakhem, F. Eassa, F. Alsolami, and A. S. A.-M. Al-Ghamdi. Technical study of deep learning in cloud computing for accurate workload prediction. *Electronics*, 12(3):650, 2023.

[4] M. Amiri and L. Mohammad-Khanli. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, 82:93–113, 2017.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[6] K. L. Devi and S. Valli. Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing*, 105:353–374, 2023.

[7] Hitesh Dhake, Yash Kashyap, and Panos Kosmopoulos. Algorithms for hyperparameter tuning of lstms for time series forecasting. *Remote Sensing*, 15(8), 2023.

[8] Y. Einav. Amazon found every 100ms of latency cost them 1% in sales. `https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/`, 2019. Accessed: 2024-08-22.

[9] Jiechao Gao, Haoyu Wang, and Haiying Shen. Machine learning based workload prediction in cloud computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2020.

[10] Shankha Shubhra Gupta. Federated learning for traffic forecast in cloud infrastructure (project proposal). Submitted as part of the requirements for CE902 Professional Practice and Research Methodology, under the supervision of Mays Al-naday, March 2024.

[11] Fady Hassouna. What is the problem with using r-squared in time series models?, 07 2020.

[12] S. L. Ho, M. Xie, and T. N. Goh. A comparative study of neural network and box-jenkins arima modeling in time series prediction. *Computers & Industrial Engineering*, 42(2):371–375, 2002.

[13] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi. Machine learning-based auto-scaling for containerized applications. *Neural Computing and Applications*, 32:9745–9760, 2020.

[14] Vinodh Kumaran Jayakumar, Jaewoo Lee, In Kee Kim, and Wei Wang. A self-optimized generic workload prediction framework for cloud computing. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 779–788, 2020.

[15] Cao Jian, Fu Jiwen, Li Minglu, and Chen Jinjun. Cpu load prediction for cloud environment based on a dynamic ensemble model. *Software: Practice and Experience*, 44(7):793–804, 2014.

[16] Nan Deng Md Ehtesam Haque John Wilkes, with much help from Charles Reiss and Muhammad Tirmazi. Google cluster-usage traces v3. Original version 2020-04-01, updated 2020-05-01, 2020-07-28, 2020-08-10, 2020-08-18.

[17] İ. Kalafat, M. Hekimoğlu, A. D. Yücekaya, N. Ay, and H. Gültekin. Workload forecasting of warehouse stations: Comparison between classical time series methods and xgboost. *International Journal of Data Science and Applications*, 4(2):19–24, 2021.

[18] Tahseen Khan, Wenhong Tian, Guangyao Zhou, Shashikant Ilager, Mingming Gong, and Rajkumar Buyya. Machine learning (ml)-centric resource management in cloud computing: A review and future directions. *Journal of Network and Computer Applications*, 204:103405, 2022.

[19] Vladislav Kramar and Viktor Alchakov. Time-series forecasting of seasonal data using machine learning methods. *Algorithms*, 16(5):248, 2023.

[20] J. Kumar and A. K. Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41–52, 2018.

[21] Ji Liu, Juncheng Jia, Beichen Ma, Chendi Zhou, Jingbo Zhou, Yang Zhou, Huaiyu Dai, and Dejing Dou. Multi-job intelligent scheduling with cross-device federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 34(2):535–551, 2023.

[22] Yi Liu, Shuyu Zhang, Chenhan Zhang, and James J.Q. Yu. Fedgru: Privacy-preserving traffic flow prediction via federated learning. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020.

[23] M. Masdari and A. Khoshnevis. A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing*, 23:2399–2424, 2020.

[24] W. Mensi, X. V. Vo, and S. H. Kang. Upside-downside multifractality and efficiency of green bonds: The roles of global factors and covid-19. *Finance Research Letters*, 43:101995, 2021.

[25] V. R. Messias, J. C. Estrella, R. Ehlers, M. J. Santana, and H. Senger. Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Neural Computing and Applications*, 27:2383–2406, 2016.

[26] Rizwan Mushtaq. Augmented dickey fuller test. Available at SSRN: `https://ssrn.com/abstract=1911068` or `http://dx.doi.org/10.2139/ssrn.1911068`, 2011. Accessed: 2024-08-22.

[27] Seol-Hyun Noh. Analysis of gradient vanishing of rnns and performance comparison. *Information*, 12(11):442, 2021.

[28] Marco Piacentini and Francesco Rinaldi. Path loss prediction in urban environment using learning machines and dimensionality reduction techniques. *Computational Management Science*, 8(4):371–385, 2011.

[29] John J. Prevost, KranthiManoj Nagothu, Brian Kelley, and Mo Jamshidi. Prediction of cloud data center networks loads using stochastic and neural models. In *2011 6th International Conference on System of Systems Engineering*, pages 276–281, 2011.

[30] Víctor Rampérez Martín. A technology-agnostic approach to auto-scale services in heterogeneous clouds. Unpublished, May 2021.

[31] M. Rareshide. Power in the data center and its cost across the u.s. `https://info.siteselectiongroup.com/blog/power-in-the-data-center-and-its-costs-across-the-united-states`, 2017. Accessed: 2024-08-22.

[32] Deepika Saxena and Ashutosh Kumar Singh. workload forecasting and resource management models based on machine learning for cloud computing environments. *CoRR*, abs/2106.15112, 2021.

[33] Prince Sekwatlakwatla, Maredi Mphahlele, and Tranos Zuva. Traffic flow prediction in cloud computing. In *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pages 123–128. IEEE, 2016.

[34] K. Seshadri, C. Pavana, K. Sindhu, and C. Kollengode. Unsupervised modeling of workloads as an enabler for supervised ensemble-based prediction of resource demands on a cloud. In P. Verma, C. Charan, X. Fernando, and S. Ganesan, editors, *Advances in Data Computing, Communication and Security*, volume 106 of *Lecture Notes on Data Engineering and Communications Technologies*, pages 125–137. Springer, Singapore, 2022.

[35] Uphar Singh, Sanghmitra Tamrakar, Kumar Saurabh, Ranjana Vyas, and O.P. Vyas. Hyperparameter tuning for lstm and arima time series model: A comparative study. In *2023 IEEE 4th Annual Flagship India Council International Subsections Conference (INDISCON)*, pages 1–6, 2023.

[36] Amit M. Thombre. Using machine learning to find out when to use box-cox transformation on time series data. In *2018 IEEE Punecon*, pages 1–7, 2018.

[37] Carlos Velasco. Estimation of time series models using residuals dependence measures. *The Annals of Statistics*, 50(5):3039–3063, 2022.

[38] Jules Waku, Kayode Oshinubi, Umar Muhammad Adam, and Jacques Demongeot. Forecasting the endemic/epidemic transition in covid-19 in some countries: Influence of the vaccination. *Diseases*, 11(4):135, 2023.

[39] Yan Wang and Yuankai Guo. Forecasting method of stock market volatility in time series data based on mixed model of arima and xgboost. *China Communications*, 17(3):205–221, 2020.

[40] Gary White and Siobhán Clarke. Short-term qos forecasting at the edge for reliable service applications. *IEEE Transactions on Services Computing*, 15(2):1089–1102, 2022.

[41] Patrick Willems. A time series tool to support the multi-criteria performance evaluation of rainfall-runoff models. *Environmental Modelling & Software*, 24(3):311–321, 2009.

[42] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1:119–132, 1998.

[43] Ji Xue, Feng Yan, Robert Birke, Lydia Y. Chen, Thomas Scherer, and Evgenia Smirni. Practise: Robust prediction of data center time series. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 126–134, 2015.

[44] A. Yadav, C. K. Jha, and A. Sharan. Optimizing lstm for time series prediction in indian stock market. *Procedia Computer Science*, 167:2091–2100, 2020.

[45] Mahendra Pratap Yadav, Nisha Pal, and Dharmendar Kumar Yadav. Workload prediction over cloud server using time series data. In *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 267–272, 2021.

[46] H. Yang, H. Wang, Y. Gao, X. Liu, and M. Xu. A significant wave height forecast framework with end-to-end dynamic modeling and lag features length optimization. *Ocean Engineering*, 266:113037, 2022.

[47] Yongjia Yu, Vasu Jindal, Farokh Bastani, Fang Li, and I-Ling Yen. Improving the smartness of cloud management via machine learning based workload prediction. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 02, pages 38–44, 2018.

[48] Yongjia Yu, Vasu Jindal, I-Ling Yen, and Farokh Bastani. Integrating clustering and learning for improved workload prediction in the cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 876–879, 2016.

[49] L. Zhang, W. Bian, W. Qu, L. Tuo, and Y. Wang. Time series forecast of sales volume based on xgboost. In *Journal of Physics: Conference Series*, volume 1873, page 012067. IOP Publishing, 2021.

[50] Qi Zhang and Raouf Boutaba. Dynamic workload management in heterogeneous cloud computing environments. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–7, 2014.

[51] Eric Zivot and Jiahui Wang. Rolling analysis of time series. In *Modeling Financial Time Series with S-Plus®*, pages 201–244. Springer, New York, NY, 2003.

Appendix

The project repository can be accessed by clicking the following link:

Click here to view the project repository.