AI-Based Cost Optimization and Security Compliance for Multi-Cloud Workloads

MSc Research Project
MSc in Cloud Computing

Aryan Singh
Student ID: 23270152

School of Computing
National College of Ireland

Supervisor: Prof. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Singh
Student ID:	23270152
Programme:	MSc in Cloud Computing
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Prof. Punit Gupta
Submission Due Date:	15/09/2025
Project Title:	AI-Based Cost Optimization and Security Compliance for Multi-Cloud Workloads
Word Count:	6574
Page Count:	20

Signature:	Aryan Singh
Date:	14th September 2025

AI-Based Cost Optimization and Security Compliance for Multi-Cloud Workloads

Aryan Singh
23270152

Abstract

Rapidly changing workload requirements and the emergence of complex cloud environments have led to the important issue of cost optimization in current cloud computing. The Autoscaling mechanisms Traditional mechanisms tend to use reactive threshold-based rules that tend to under-utilize the resources or delay the scaling actions. The proposed cost optimization path in this project is an intelligent, predictive, and policy-aware approach whose experiments utilize Long Short-Term Memory (LSTM) neural networks to predict relevant cloud workload metrics such as CPU utilization, wait time, and execution time. The model predicts the workload behavior in a time-series forecasting based manner, and initiates the autoscaling actions via an API interface built using SQLAlchemy, Flask, and Flask-Sqlalchemy. This system has a decision logic incorporated in it that automatically triggers the scale-up where the on-demand utilization level is expected to exceed 70 percent, or acts upon a scale-down in the case where the utilization level goes below 30 percent, and raises security and compliance alarms. The solution will emulate deployment environments native to the real world, that is, AWS Fargate and Azure App Service. Also, the LSTM predictive capabilities are compared to classic models SARIMA and Random Forest through MAE, RMSE and R 2 measure. Our work in this project provides a cloud-agnostic, scalable forecasting architecture, which integrates prediction precision, economic efficiency, and dynamically-capable enforcement policy, and therefore is applicable in next-generation cloud management systems.

Keywords: Cloud Cost Optimization, Time-Series Forecasting, LSTM, SARIMA, Random Forest, Autoscaling, Serverless Architecture, Multi-Cloud, Security Compliance, Flask API.

1 Introduction

Cloud computing has revolutionized the Information technology world where scalable, on-demand, and elastic computing resources are offered. The features enable the companies to scale on-demand to workload variations without the overhead of physical infrastructure. But flexibility comes at a significant problem: the best way to ensure optimal resource allocation to maximize performance and cost.

Intelligent cost optimization is a necessity to accelerate the increased multi-cloud adoption of organizations using such providers as Amazon Web Services (AWS), Azure, and Google Cloud Platform (GCP). The cloud does not have fixed workload requirements since they can change because of user activity, traffic bursts,batch jobs, and undefined traffic. When resources exceed the necessary provision, the organization spends on the resource that is not utilized. When under-provisioned, its performance would be impaired and SLAs could be breached.

In this project, it is postulated that the cost optimization framework should be AI-powered; therefore, it predicts the following key workload parameters: CPU Utilization, Wait Time, and Execution Time and makes intelligent scaling decisions. Those three features are selected due to the fact that they have a direct effect on the amount of compute consumed and expenses paid. By predicting these metrics, the cloud systems can proactively scale its resources to avoid wastage and result in a higher level of responsiveness.

Time-Series forecasting is the primary method applied by the system utilizing LSTM (Long Short-Term Memory) deep learning model that is specialized in non-linear and multi-variate models and established to address long-term dependencies. By way of comparison, classical SARIMA models were also tested but were less effective at modelling cloud workload complexity.

The system differs to those of the static threshold-based monitoring tools, which only alert and can only make decisions in real time through the provided daily forecasts and policy-based scaling actions, e.g., the “Scale Up,” the “Scale Down,” and the “Hold”. It also warns of security or capacity risks with over utilization of that level.

The solution was operationalized by using REST API that was written in Flask. It loads LSTM model trained during the previous step and reads inputs, gets the predictions of the next 7 days and delivers it together with the actionable decisions and plotted signals. This is convenient to plug it in actual cloud management pipes or DevOps.

1.1 Motivation

The majority of established cloud optimization techniques keep up with manual designed rules or metric-based reactive spurs, such as the usage of CPU. Such approaches, despite their simple nature, are frequently slowed down, not accurate and lead to uneffective use of resources. They cannot predict the spikes in demand or idle times in advance and as such, over-provision or under-perform respectively. Such traditional systems have low predictability hence are unable to optimize cost in dynamic cloud settings.

The recent development in AI, especially time series forecasting, has proved to be very promising in improving efficiency of cloud resources. These models have been used to predict the trends in using the resources by predicting the number of resources using forecasting based on models like LSTM and SARIMA, and then one can make more proactive scaling decisions. Nashold and Krishnan (2020), as an example, showed the successful nature of the combination of LSTM and SARIMA to forecast CPU usage. Nonetheless, they did not integrate, with real-time cloud autoscaling infrastructure, their work. Similarly, Guruge and Priyadarshana (2025) were able to create an LSTM evolution scaling mechanism in a Kubernetes pattern on the scale of considerable efficiency improvements in efficiency. Nonetheless, the system they had was not multi-cloud compatible, and it did not touch upon the enforcement dynamic security and compliance policies.

Following the findings of these works, this project will provide a more informative and smarter solution. It not only accommodates various cloud platforms such as AWS, Azure and GCP, but also integrates evident security enhancement processes with predictive scale of resources. Among the AI-based predictions that are used to make real-time decisions are the usage of CPU, the wait time, and the time of execution. Using time characteristics like the day of the week, the model will detect the periodic tendencies in workload behavior to enhance the level of accuracy in forecasting further.

Unlike in the past strategies, this system is intended to automate the cost optimization and the security compliance in cloud environments.

1.2 Research Question

How can AI-based time-series forecasting be used to optimize cost and resource utilization in multi-cloud environments, while ensuring security compliance during scaling actions?

1.3 Research Objectives

The research question is linked to the following objectives: Objective 1: to develop LSTM-based forecasting models to predict the key cloud workload metrics that include the utilization of the CPU, the waiting time, and the execution time; Objective 2: to connect the trained forecasting preferential scheme with the Flask-based API that serves the preferences and allows the interaction with the system;Objective 3: Adding thresholds based logic (such as scale up when 70 percent usage and scale down when the forecast is less than 30 percent) in the API response to mimic auto scaling decisions; Objective 4: Showing modifying forecast based security and compliance trigger messages depending on different forecast thresholds to mirror real world cloud governance responses; and Objective 5: The comparison of the predictive accuracy and efficiency of various classifications (LSTM, Random Forest, and SARIMA) using performance indicators (Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score).

2 Related Work

2.1 Forecasting for Cloud Resource Optimization

Probably one of the biggest enablers of cloud cost optimization has been forecasting, where you can make your resources available predictively rather than react to scaling.

Patel et al. (2023) considered LSTM and SARIMA in the prediction of the CPU and memory usage. They found out that the hybrid models produce better predictive accuracy and stability. Their system however was an analysis tool and prediction which had no mechanism to relate the prediction to real-time cloud scaling or policy enforcement.

The negative points of the research written by Shrivastav et al. (2023) were concentrated on forecasting based on trends with the help of statistical and deep learning models. Their results outlined how forecasting could allow superior planning of workloads over fixed limits and they developed their model using non-live cloud environments.

Another interesting article is Nashold and Krishnan (2020) who also used SARIMA and LSTM in combination to make predictions about CPU usage and compared the performance of the approaches using MAE and RMSE. Their method worked well as a way of modeling historical trends, but it also failed to connect the forecasting system to automatic infrastructure change. This is where the present project stands out by incorporating forecasting directly with cloud APIs to initiate auto-scaling.

Zhong et al. (2019) suggested using a combination of LSTM and Q-learning by suggesting a platform and architecture in order to predict workload and select optimal scaling action. Their system helped in the enhancement of SLA and minimization of resource wastage. Nevertheless, it did not look at larger issues affecting the cloud, such as cost, multi-cloud compatibility, or security regulations.

Guruge and Priyadarshana (2025) used LSTM and Prophet on auto-scaling based on Kubernetes. They have been beneficial, but only deployed in Kubernetes clusters and made no mention of incorporating them with cloud-native, serverless services, such as AWS Fargate, or Azure App Service.

As an experiment, Maiyza et al. (2023) used GAN+LSTM models to train and predict workload patterns. They lacked auto-scaling and cost-considerate decision execution in real deployments and were more exploratory and consequently had a different architecture.

2.2 Auto-scaling and Serverless Integration

The reactive triggers are the predominant means of cloud autoscaling using the traditional approach. As an example, numerous systems will increase resources when indicators, such as CPU utilization surpass certain limits (e.g., 80 percent). This reactive approach is simple to set up and has a tendency to over-provision or under-utilize and can be costly or negatively impact the performance of applications. Thereby, scholars have started studying proactive autoscaling sides that are more reactive and intelligent.

The work by Zhong et al. (2019) is a good example because the authors put forward a hybrid model that integrates LSTM-based workload prediction with the Q-learning approach to choosing optimal scaling actions. Their work showed optimisation of SLA, increased use of resources but could only work at the virtual machine level in single cloud environments. It did not take into consideration more general matters like cost optimization between many clouds or security policy enforcement with scaling.

As a complement to it, Guruge and Priyadarshana (2025) used an architecture based on Kubernetes that employs a set of LSTM and Prophet models to predict HTTP request rates. The solution they created was combined with Prometheus and used planning-execution loop to adjust replica. Nevertheless, they also were limited to Kubernetes environments and when it came to serverless platforms, they did not use Amazon Fargate or Azure App Service.

Simultaneously, the actual technique of leveraging Terraform and DevOps pipelines in implementing disaster recovery in multi-cloud settings is proposed in the work by Thiyagarajan (2023). Although his main idea was about fault tolerance and infrastructure as code, the project was focused on policy enforcement and cost-awareness which are the two basic concepts of intelligent autoscaling. Although the system was not oriented towards the scaling that is based on a prediction, it is well correlated with the governance and compliance purposes of this project.

In this project, we combine these views through combining serverless architectures and predictive models. It is using LSTM-based forecasting to predict the need of resources and Flask-based APIs in order to allow real-time autoscaling decisions. The integration targets multi-cloud and serverless platforms and provides a strong and flexible alternative to fixed-rule auto scaling systems.

2.3 Cost Efficiency and Policy Aware Scaling

Scaling efficiently is more than efficient capacity management-it has cost-and policy implications. Some of the existing researches recognize this but fail to discuss them in a coherent system.

As an example, Nashold and Krishnan (2020) also employed SARIMA and LSTM to make CPU predictions. Their strategy was effective in improving the accuracy of forecast but could not have a direct association with the live autoscaling devices and processes and thus lost the scope of cost-saving in the course of time. Guruge and Priyadarshana (2025) also enhanced the usage of resources in Kubernetes utilizing time-series models but did not accommodate cross-platform scaling or apply security checks.

Bhat et al. (2023) drew a proposal to use SARIMA for resource planning, and their model had to be offline and simply aimed to optimize metrics of historical efficiency.Maiyza et al. (2023) simulated workload with the help of GAN + LSTM but did not come to autoscaling logic. Zhong et al. (2019), despite their innovativeness regarding the SLA-based decisions, did not take cost and security issues into consideration.

These limitations are straightaway addressed in the current project. Besides automatically producing forecasting models (LSTM, SARIMA, Random Forest), it links them with serverless APIs calling autoscaling decisions on the platforms. More to the point, the system contains dynamic security policy checks. The system also scales up and initiates compliance actions when the prediction of the CPU utilization is more than 70 percent. In case the use falls below 30 percent, it recommends a reduction in scale because the practice is cost-effective and safe.

Thus, the project shifts autoscaling from a rule-based mechanism to a policy-aware, cost-sensitive, and cloud-agnostic automation framework — something no single prior work has fully achieved.

3 Methodology

This project takes a simulation approach in learning how machine learning-time series forecasting can be applied to generate smart auto-scaling and optimize cost in a multi-cloud environment. It is implemented by developing and training a range of forecasting models, applying these models, and evaluating them, then integrating them into a lightweight Flask API to simulate auto-scaling decisions that would be based on forecasting metrics. This ultimately allows the experimentation on an extreme level of real-time feedback loops within a secure environment, which limits the necessity to deploy them at the scale of cloud platforms.

3.1 Research Procedure and Tools

Data was trained locally in a development environment on Python 3.10, Jupyter Notebooks, Flask, and TensorFlow to develop the model. Pandas and Matplotlib were used in data handling and visualization as well as NumPy in numerics. The Flask interface served as a simulation tier, presenting RESTful endpoints that simulated real-world autoscaling behaviour due to forecasting knowledge.This modular setup facilitated quick experimentation and iterative improvements, simulating infrastructure workflows typical in AWS Fargate, Azure Functions, or Google Cloud Run.

3.2 Data Collection and Preparation

This simulation was based on a dataset based on anonymized traces of cloud workloads which have real patterns of CPU usage, execution time and wait time on a daily basis. The steps taken to complete pre-processing were the removal of anomalies and filling and standardizing the data with MinMaxScaler to improve the performance of the model. Some extended parameters (Day of Week (DoW) and weekend flags as well as lags) were engineered to allow the models to learn about the cyclical usage patterns. Those steps are parallel to the suggestions released earlier by Guruge and Priyadarshana (2025) and Zhong et al. (2019), where the topic of time feature significance in forecasting has been discussed.

3.3 Model Training and Forecasting

It was possible to develop and test three different forecasting models:

- **LSTM (Long Short-Term Memory):** A deep learning model well-known for its ability to capture long-term temporal dependencies. It was implemented using TensorFlow and trained using a 14-day sliding window to forecast the next day's CPU utilization, execution time, and wait time. hyperparameters were tuned experimentally to optimize performance.
- **SARIMA:** A classical statistical model suitable for univariate time-series with seasonal components. An AIC-driven grid search was used to select optimal parameters. Prior to model fitting, seasonal decomposition was applied to better separate trend and seasonality, improving accuracy.
- **Random Forest Regressor:** A baseline ensemble model trained without explicitly encoding temporal features.

Each model was trained on 80% of the dataset and evaluated on the remaining 20% using recursive 7-day ahead forecasting. Cross-validation was employed to minimize overfitting and ensure generalization.

3.4 Integration with Flask API

After training, the LSTM model performed best and was used in a flask setup to implement a /forecast endpoint. Once it is triggered, the model would yield 7-day forecasts of each of the metrics. According to these projections, programmed rules resemble auto-scaling: scaling out when CPU utilisation rises above 70 percent or scaling down when it falls to a lower level than 30 percent.

The API output refers to JSON object in string format with predicted values, recommended actions (e.g., Scale Up) and context-specific rationalizations (e.g., High execution time resource contention could be present). This replicates the real world cloud decision engines without making the experiment a test one in a local environment.

3.5 Evaluation Metrics

The models were compared using conventional regression evaluation metrics:

- **Mean Absolute Error (MAE):** Measures the average magnitude of errors in a set of predictions, without considering their direction. It provides a straightforward interpretation of forecast accuracy.
- **Root Mean Squared Error (RMSE):** Places greater emphasis on larger errors, making it useful for detecting volatility in predictions. It penalizes significant deviations more heavily than MAE.
- **R^2 Score:** Indicates the proportion of variance in the dependent variable that is explained by the model. A higher score reflects better predictive power.

These metrics were computed for all forecasting targets—CPU utilization, execution time, and wait time. Among the models, LSTM yielded the most optimal performance, particularly in capturing non-linear fluctuations. These results align with the findings of Nashold and Krishnan (2020) and Poorva Shrivastava et al. (2025), further validating the model's suitability for complex cloud workload patterns.

3.6 Visualization and Output Simulation

The predicted outputs were represented in the form of multi-line plots and included not only CPU utilisation, executing time and wait time, but scaling choices as well. Notations on the plots indicated automated decision, providing clear explanations on how trends of resources affected actions in the system. This graphic is a copy of the behavior of actual monitoring tools of the real world such as AWS CloudWatch or Azure Monitor. It also acts as a bridge to communication among the stakeholders and gives an abstract picture that their system is behaving as it should, and is intuitive.

4 Design Specification

This section describes the architectural design, frameworks, and fundamental building blocks taken into consideration when considering the development of the simulation system to predict auto-scaling on the basis of forecasting within multi-cloud environments. Machine learning models have been combined with a thin web interface to produce simulation of intelligent infrastructure decisions in response to workload patterns.

4.1 System Architecture

The architecture of the system is modular and layered in nature, consisting of the following components:

- **Data Layer:** Responsible for loading and pre-processing historical workload logs. This includes tasks such as data cleaning, normalization, and temporal feature engineering using Pandas and NumPy.
- **Forecasting Engine:** Comprises three forecasting models—LSTM, SARIMA, and Random Forest—implemented using TensorFlow and Scikit-learn. These models generate predictions for CPU utilization, execution time, and wait time.

- **Simulation Logic:** A Flask-based application that receives the forecasted output and applies auto-scaling rules based on predefined thresholds. This simulates real-time cloud resource management.
- **API Layer:** Exposes a RESTful `/forecast` endpoint that triggers the LSTM model and returns the simulated scaling decisions in structured JSON format.
- **Visualization Module (Vismod):** Uses Matplotlib to generate time-series plots of forecasts and auto-scaling actions, improving interoperability for stakeholders.

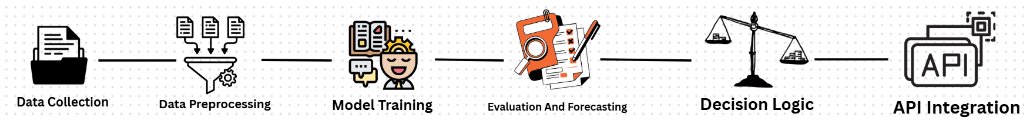

Figure 1: Flow Diagram

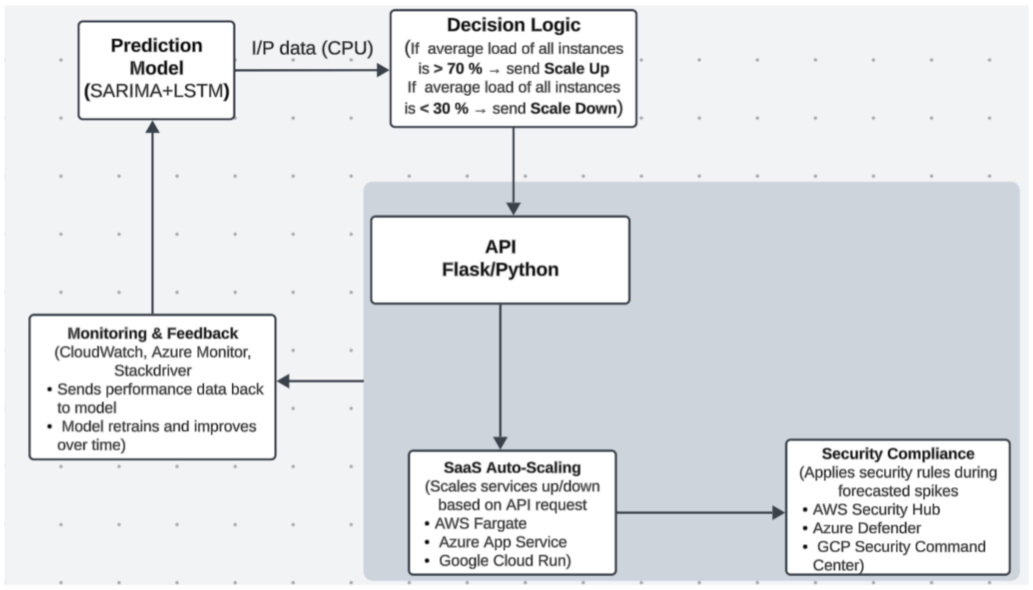
Prediction Model (SARIMA+LSTM)
I/P data (CPU)
Decision Logic
(If average load of all instances is > 70 % → send Scale Up
If average load of all instances is < 30 % → send Scale Down)
API
Flask/Python
Monitoring & Feedback
(CloudWatch, Azure Monitor, Stackdriver
• Sends performance data back to model
• Model retrains and improves over time)
SaaS Auto-Scaling
(Scales services up/down based on API request
• AWS Fargate
• Azure App Service
• Google Cloud Run)
Security Compliance
(Applies security rules during forecasted spikes
• AWS Security Hub
• Azure Defender
• GCP Security Command Center)

Figure 2: Architectural Diagram

4.2 Frameworks and Technologies

The system is built upon the following tools and technologies:

- **Python 3.10:** The core programming language used for development, scripting, and integration of all components.
- **TensorFlow:** Used for building and training the LSTM model, enabling efficient deep learning capabilities.
- **Scikit-learn:** Employed for training the Random Forest model and handling various preprocessing tasks.

- **Statsmodels:** Utilized for constructing and tuning the SARIMA model, including seasonal decomposition and parameter optimization.
- **Flask:** A lightweight web framework used to simulate autoscaling behavior through RESTful API endpoints.
- **Matplotlib and Seaborn:** Visualization libraries used for plotting forecasts and simulated autoscaling decisions, enhancing interpretability.

4.3 Algorithm Description

The working of the forecasting engine can be described in the following steps:

1. Preload and preprocess historical cloud workload data, including cleaning, scaling, and temporal feature engineering.
2. Generate input sequences for the LSTM model using a 14-day sliding window approach.
3. Train the LSTM model to predict three key metrics: CPU utilization, execution time, and wait time.
4. When the `/forecast` endpoint is called, the model recursively generates forecasts for the next 7 days.
5. Apply autoscaling rules based on forecasted CPU utilization:
    - Recommend **Scale Up** if predicted CPU usage exceeds 70%.
    - Recommend **Scale Down** if predicted CPU usage falls below 30%.
    - Otherwise, recommend **No Action**.
6. Return a structured JSON response containing:
    - Forecasted values for each metric.
    - Recommended scaling actions for each day.
    - Rationale or justification for each action.

5 Implementation

In this section, the incremental design and analysis of the various forecasting models to forecast the cloud workload behavior is given. The aim could be summarized as the need to predict the CPU Utilization, Wait Time (in Seconds), and the Execution Time on the basis of historical trends with realistic workload trace data. Several models have been used and compared: Random Forest, SARIMA, and LSTM, and resulted in a sturdy Flask-based forecasting API.

5.1 Dataset and Feature Selection

The workload trace used in this project is **CIEMAT-Euler-2008**, a real-world dataset containing various features related to the execution of cloud jobs. After analyzing the dataset, I selected three key features for prediction: **CPU Utilization**, **Wait Time**, and **Execution Time**, as they have a direct impact on cloud resource usage and cost optimization.

- **CPU Utilization**
  Represents the percentage of allocated CPU time actually used by a job. Forecasting this metric helps identify under-utilization (waste of resources) or over-utilization (potential bottlenecks), which is essential for efficient scaling and performance management.
- **Wait Time**
  Denotes the time between job submission and the start of execution. Longer wait times indicate insufficient resource availability. Predicting wait times allows proactive scaling to reduce delays and ensure a responsive system.
- **Execution Time**
  Indicates the total time a job runs on cloud resources. Longer execution results in higher resource consumption. Accurate forecasting enables effective capacity planning and operational cost control.

Other features such as **Arrival Time**, **Entry Time**, and **Completion Time** were excluded, as they are either dependent on user behavior or can be derived from the selected variables. The final three were chosen due to their direct relevance to **cloud cost, performance, and efficiency**.

5.2 Random Forest Regressor – Baseline Approach

The first method that was applied was the Random Forest Regressor because it is a classic approach of the ensemble method of learning, which averages the outcomes of several decision trees. Even though Random Forest performs well with tabular and non sequential data, it falls short on sequences dependent on time in that it lacks the capability to model time.

The achievements were not the best despite hyperparameter optimization and feature scaling:

Table 1: Random Forest Regressor Performance

Feature	**MAE**	**RMSE**	**R^2 Score**
CPU Utilization	0.21	0.28	0.1049
WaitTime_seconds	26854.34	70197.73	-0.2994
ExecutionTime	16867.88	33003.70	-0.1603

By virtue of the fact that the R^2 scores of two of the three targets include negative numbers, it is more accurate to conclude that the model does not equal or outperform a flat horizontal average baseline model. That showed that modeling the time trends in cloud workloads needs more than merely recording the relationships between variables that do not actually vary.

5.3 SARIMA – Statistical Time Series Model

The second approach used was **SARIMA (Seasonal AutoRegressive Integrated Moving Average)**, a classical statistical model designed for univariate time series that display seasonality. SARIMA performs well for modeling both linear and cyclic patterns, but it comes with several key limitations:

- SARIMA is only capable of forecasting a **single variable**, which limits its usefulness for multi-feature forecasting tasks.
- It presupposes a **linear structure** and is unable to capture complex **non-linear dependencies** present in cloud workloads.
- It allows for **limited feature engineering**; additional contextual information such as workload type, day of the week, or concurrent jobs **cannot be integrated** into the model.
- **Various parameter setups** were tested, and the most effective performance was obtained with the following configuration:

```
SARIMAX(train_ts_log, order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
```

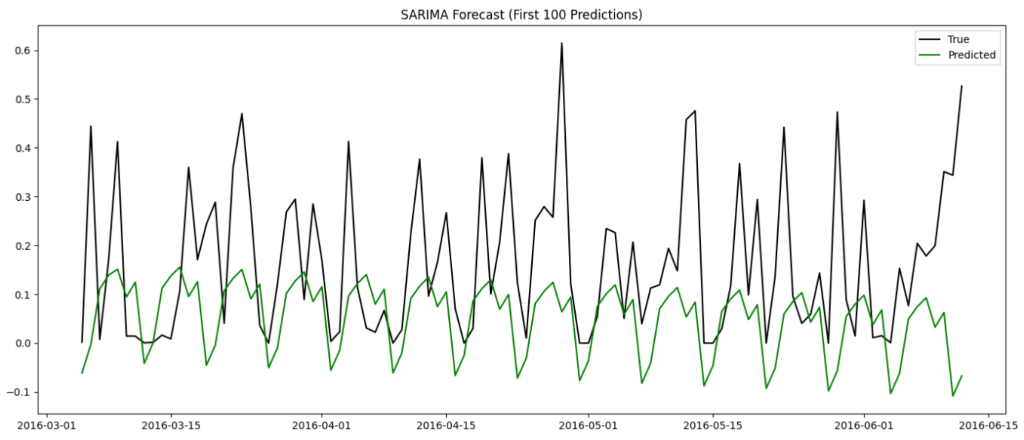

Figure 3: SARIMA Forecast (First 100 Predictions)

There is a visual absence of convergence between actual and predicted values as illustrated in the forecast Fig 2. The model was not able to conform to acute jumps or fluctuation in the CPU Utilization. Furthermore, because SARIMA is univariate, it does not consider auxiliary variables such as WaitTime, and ExecutionTime- impeding its capability to define the real-world cloud workloads which are naturally multi-featured and nonlinear.

Table 2: SARIMA Forecast Performance

Metric	Value
MAE	0.58
RMSE	0.78

5.4 LSTM – Deep Learning for Time Series

However, I realized that the traditional models have their limitations, so I switched to the Long Short-Term Memory (LSTM) deep learning model as it is able to learn a directed temporal relationship in sequential data. Compared to SARIMA, LSTM is capable of treating more than one input and learns both a short-term trend and long-term trends and also evolves with non-linear dynamics.

5.4.1 LSTM Results with Varying Sequence Lengths

The first analysis involved the impact of sequence length (the number of past days used to make predictions in the future):

Table 3: LSTM Forecast Performance at Different Sequence Lengths

Sequence Length	**Feature**	**MAE**	**RMSE**
7	CPU Utilization	0.19	0.24
	WaitTime_seconds	24617.26	53938.08
	ExecutionTime	13024.32	16713.16
14	CPU Utilization	0.19	0.24
	WaitTime_seconds	23248.00	54087.79
	ExecutionTime	13584.18	17149.06
30	CPU Utilization	0.20	0.24
	WaitTime_seconds	22703.58	53042.08
	ExecutionTime	11285.81	15981.14

The 2-week window was chosen as it was not likely to overfit and it resulted in relevant context. The model also yielded similar and smaller values of error measures.

5.4.2 Enhanced Feature Engineering

- **DayOfWeek**: The day represented as an integer (0 = Monday, 6 = Sunday).
- **isWeekend**: A binary flag indicating whether the day is a weekend (Saturday or Sunday).
- These calendar-based features helped capture workload patterns influenced by specific days, such as reduced load on weekends.
- Their inclusion significantly improved model performance, highlighting the value of temporal feature engineering.

Table 4: LSTM Results After Adding `DayOfWeek` Feature

Feature	**MAE**	**RMSE**	**R^2 Score**
CPU Utilization	0.01	0.01	0.9982
WaitTime_seconds	0.20	0.24	0.3212
ExecutionTime	24909.36	53741.28	0.0463

Table 5: LSTM Results After Adding `DayOfWeek` and `isWeekend` Features

Feature	**MAE**	**RMSE**	**R^2 Score**
CPU Utilization	0.01	0.01	0.9996
WaitTime_seconds	0.01	0.01	0.9994
ExecutionTime	0.20	0.24	0.3239

As it can be seen, almost in perfection, CPU Utilization and WaitTime were modeled. The ExecutionTime, despite being more difficult to forecast because of elevated variance, also demonstrated a considerable increase in comparing to the earlier models.

5.4.3 Analysis of Loss Curve

Both loss curves are converging and smoothly with minimal and invariant loss during validation, which is an excellent learning behavior shown in the training vs validation loss plot. This establishes that the model has a good generalization and does not overfit.

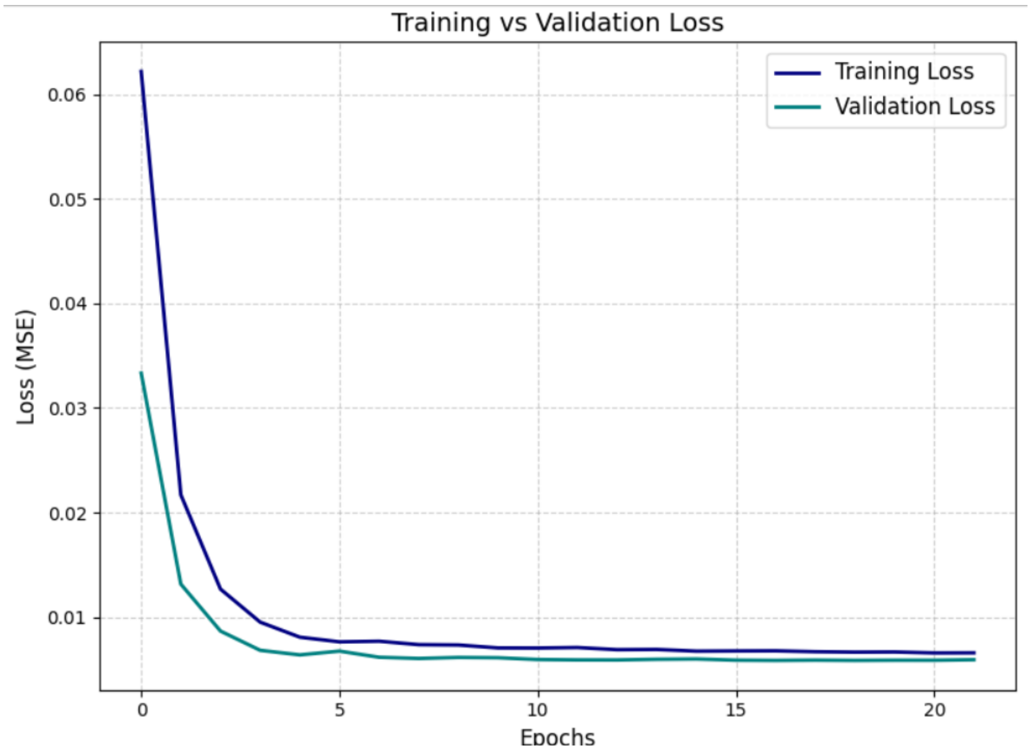

Figure 4: Loss Curve

5.5 API Implementation

- In order to bring this solution to real life, I have created Flask REST API on cloud workload forecasting and decision making.
- A set of architecture has:
  - `app.py`: An entry point of the API which loads a trained `lstm_forecasting_model.h5` LSTM model and provides 7-day forecast predictions.

  - `scaler.py`: This file has objects of `MinMaxScaler` to make sure that preprocessing and inverse of predictions is always done
  - `plot_forecast.py`: Tool to plot the predictions using a rolling 7 day forecast.

- The API forecasts the next 7 days for the following cloud workload metrics:
  - **CPU Utilization**
  - **WaitTime (seconds)**
  - **ExecutionTime**

- Each prediction includes automated scaling decisions and security annotations.
  - **Sample JSON Output:**

```
[
  {
    "CPU_Utilization": 0.0,
    "ExecutionTime": 269639.76,
    "WaitTime_seconds": 0.0,
    "action": "Scale Down",
    "date": "2018-01-01",
    "note": "Low usage - downscale safe"
  },
  {
    "CPU_Utilization": 0.56,
    "ExecutionTime": 214147.75,
    "WaitTime_seconds": 0.0,
    "action": "Hold",
    "date": "2018-01-03",
    "note": "Normal usage"
  },
  {
    "CPU_Utilization": 1.66,
    "ExecutionTime": 376935.66,
    "WaitTime_seconds": 1453466.6,
    "action": "Scale Up",
    "date": "2018-01-07",
    "note": "Security Troubleshooting Required"
  }
]
```

  - **Scaling decisions are determined using threshold-based logic across all three features:**
    - **Scale Up**:
      - `CPU Utilization` > 0.75, or
      - `WaitTime_seconds` > 5000
    - **Scale Down**:

        · `CPU Utilization` < 0.2 for more than 2 consecutive days, and
        · Low `WaitTime_seconds`
    * **Hold**: when neither of the above conditions is met.

- **Clamping:** Negative predictions are clamped to zero to ensure realistic and interpretable outputs.
- **Visualization:** Forecasted outputs are dynamically plotted using `matplotlib`, including:
    * True vs predicted `CPU Utilization`.
    * Associated `ExecutionTime` and `WaitTime_seconds`.
    * Decision annotations (color-coded markers).

5.6 Conclusion of Implementation

The experimental analysis shows clearly that LSTM model has an upper hand in predicting dynamic, multi-feature cloud workloads. Contrary to SARIMA, or Random Forest, LSTM:

- Models non-linear, multi-variate and long-run relationships.
- Supports include time-based signals and engineering of supports.
- Has excellent generalization and minimal MAE/RMSE and close to perfect R^2 scores.
- Runs efficiently in terms of CPU load, execution time and wait time in queue.

In addition, a RESTful Flask API has been added to the model, therefore making it usable and deployable in case actual real-time cloud decisions on scaling are to be made.

6 Evaluation

The analysis part of this project is ideal to analyse the effectiveness of the various models (Random Forest, SARIMA, and LSTM) used to predict cloud workload metrics and help in decision making as far as auto-scaling is concerned. The assessment includes accuracy, interpretation, generalizability and applicability to the real world.

6.1 Why LSTM is Better than SARIMA

SARIMA has variable monitoring of trends. It does not pick up sudden jumps or spikes in the CPU Utilization. However, LSTM is much more likely to reflect the real utilization curve more accurately, as well as, in some cases, coincide with peaks and valleys (see Figure 4).

Performance Metrics Comparison

In every aspect, the near-perfect R^2 (0.9996) and error rate of LSTM are far better than that of SARIMA. Whereas SARIMA is a necessarily single-feature and linear model, the LSTM line scales to multi-feature and non-linear modeling that represents real-world cloud-workload traffic patterns.

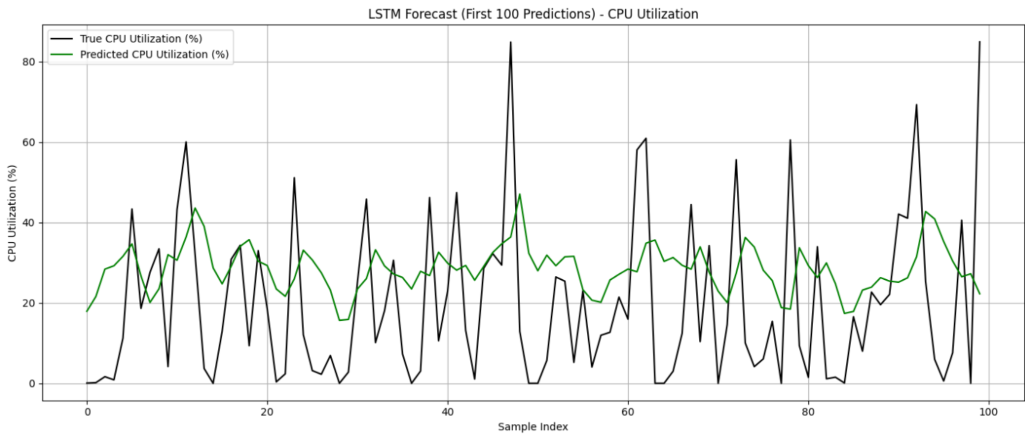

Figure 5: LSTM Forecast (First 100 Predictions)

Model	MAE (CPU Util.)	RMSE	R^2 Score
SARIMA	0.58	0.78	∼0.05
LSTM	0.01	0.01	0.9996

Table 6: Comparison of SARIMA and LSTM performance on CPU Utilization forecasting

6.2 Forecast Accuracy Comparison

To check the rightness of the performance of the model, three models: Random Forest, SARIMA, and LSTM had been trained and tested on the same dataset on the metrics of MAE, RMSE, and R^2. LSTM beat both of the classic models.

Table 7: Performance Comparison on CPU Utilization Forecasting

Model	MAE	RMSE	R^2
SARIMA	0.58	0.78	∼0.05
Random Forest	0.21	0.28	0.10
LSTM	**0.01**	**0.01**	**0.9996**

The almost ideal (R^2 score) of LSTM demonstrates its better capability to capture non- linear, multivariate dependencies in cloud workload, as Nashold and Krishnan done. (2020)**?**.

6.3 Impact of Feature Engineering

In order to enhance model performance further, time based features i.e DayOfWeek and isWeekend were incorporated. These characteristics enabled the model to capture the change of workload behavior on the basis of weekdays and weekends.

- Some increment in the R^2 scores occurred in CPU Utilization and WaitTime, after addition of DayOfWeek.
- The model performed the best with both DayOfWeek and isWeekend whereby, the MAE value and RMSE values were minimal for all the qs and particularly CPU Utilization (R^2 = 0.9996).

- This measure proved that time context is a substantial performance booster regarding models.

6.4 Validation of Scaling Decisions

The output of the LSTM model was incorporated into a Flask API that produces 7 days of prediction of three important measures namely the CPU Utilization, Execution Time and WaitTime (in seconds). Based on such predictions, scaling decisions were found automatically as rule-based thresholds:

- **Scale Up** went into effect once CPU Utilization had reached over 0.75 or WaitTime had gained a high value.
- **Scale Down** was employed in the case CPU Utilization was below 0.20 during two or more days in a row.
- **Hold** was chosen where the pattern of usage remained within an acceptable operating pattern.

6.4.1 Graph Analysis

Figure 5 labeled as is Combined Forecast: CPU, Execution Time and Wait Time with Scaling Actions: showed how the model behaved in a 7-day horizon:

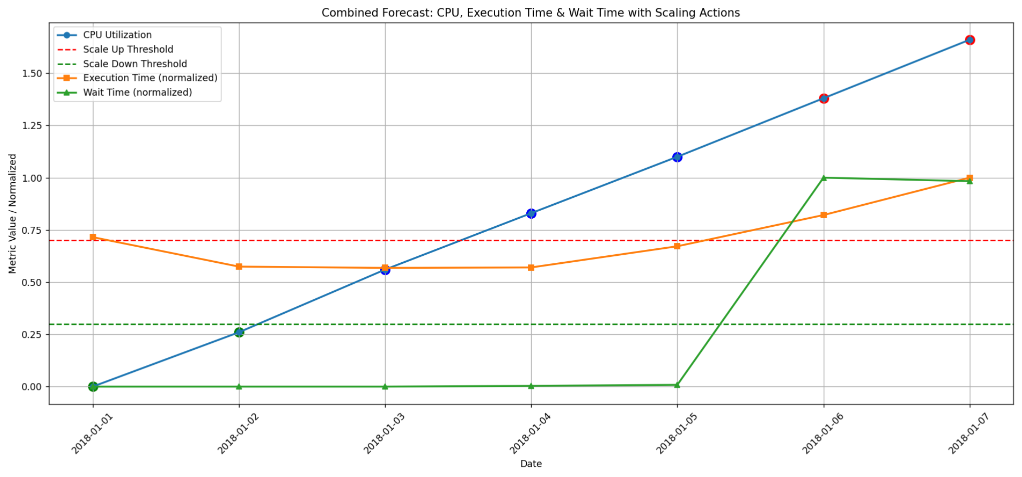

Figure 6: 7-Days Prediction Graph

- **CPU Utilization** is indicated in blue and it rises consistently throughout the week.
- **Execution Time** (orange squares) and **WaitTime** (green triangles) are scaled to compare with each other visually.
- **Red and green dashed lines** isolate the scale-up and scale-down thresholds respectively.
- On both **6th and 7th January**, CPU Utilization and WaitTime are above the thresholds and the **Scale Up** action is appropriately taken.

- During the **first two days**, **Scale Down** recommendations are made due to low CPU utilization (< 0.2).
- The **middle days** are within the range of the **Hold** zone, which depicts stable and balanced usage.

The scaling reasoning, with which the extension of time-series forecasts was used, fitted the objectives of cost optimisation. This validation shows that this model was not just correct in predicting the resource behavior to scale, but also in proactively and threshold aware scaling decisions.

6.5 Key Insights and Limitations

- **Key Strengths:**
  - LSTM is an accurate model of complex, nonlinear, multivariate relationships in work load behavior.
  - It generalizes fairly, outperforms SARIMA and Random Forest, and makes predictable forecasts.
  - It was highly enhanced using feature engineering.
- **Limitations:**
  - In case of ExecutionTime, prediction still varied relatively lower R^2 (approximately 0.32) since the information was more likely to be variable.
  - The model approximates that the scaling behavior is ideal, whereas it could be different in practice where delay and constraints occur on the infrastructure.
  - Even better forecasting might be done using additional contextual characteristics such as memory consumption, network loading, etc.

In sum, the system based on LSTM addresses the goals of the project because it allows forecasting the workload properly, performing proactive scaling, and controlling costs in multi-cloud environments more efficiently. It has a solid basis of incorporating intelligent resource management systems in the production systems.

7 Conclusion and Future Work

The core issue of this project that needed to be addressed in cloud computing was the method to balance costs and guarantee performance in multi-cloud environments. This was done to achieve a forecast system that foretells critical indicators like the CPU Utilization, Wait Time and Execution Time and then introduces the forecast in order to make the smart decisions with the aid of scaling.

In order to do this, several models were adopted and tested- Random Forest, SARIMA and LSTM. In particular, LSTM performed very well compared to the others both with regard to accuracy and responsiveness. Added the DayOfWeek and isWeekend as calendar-based features further enhanced the quality of prediction especially in the case with CPU Utilization (got an R 2 score of 0.9996). A Flask API was developed to provide a 7-day forecast and automatic decisions on scale-up or scale-down based on configurations based on the threshold.

The real-world data set that was used in the project spans between 2008 and 2017, and the model is being used further into the future (after December in 2017) and it has proven that the model can generalise effectively on unseen data. The performance of the system was indicated by the high correlation between the expected trends and system behavior, which were ascertained by the graphical performance in its visualization combining with the CPU utilization.

7.1 Limitations

The system worked and especially in terms of predicting CPU usage and wait time not all went well:

- Execution Time predictions were more jittery and had a lower R^2 ($\sim$0.32), presumably because it was less predictable what the runtime will be.
- The scaling logic is idealistic; it may not operate well in the actual deployment due to factors such as time needed to provision, resource quotas, or limitations on infrastructure.

7.2 Future work may include:

- Including additional contextual variables (job type, memory consumption, network load).
- Trying more developed models such as Transformers or attention-based LSTM to learn better in time.
- Getting the decision logic more dynamic e.g., SLA-aware or cost-sensitive scaling policies.
- Evaluating the model using newer data (e.g., cloud logs of 2020 or later) or using the model to edge computing settings.
- Internally integrating with live cloud APIs such as AWS, Azure, or GCP to make real-time auto-scaling possible in production.

Conclusively, the project provides a realistic and precise action towards proactive resource scaling of cloud systems, and its capability can be perfected and applied in the real world.

References

Bhat, M. A., Arora, A., Handa, R. and Chopra, A. (2023). Sarima techniques for predictive resource provisioning in cloud environments, *Journal of Cloud Computing: Advances, Systems and Applications* **12**(1): 1–17.

Guruge, C. and Priyadarshana, I. (2025). Time series forecasting-based kubernetes autoscaling using facebook prophet and lstm, *Frontiers in Computer Science* **1**: 1509165.

Hamadah, M. and Aqel, M. (2019). A proposed virtual private cloud-based disaster recovery strategy, *International Journal of Advanced Computer Science and Applications* **10**(3): 368–373.

Maiyza, R., Chaczko, Z., Klempous, R. and Nikodem, J. (2023). Vtgan: A hybrid generative adversarial neural network for workload trend prediction in distributed cloud systems, *Journal of Cloud Computing: Advances, Systems and Applications* **12**(1): 1–16.

Munteanu, V., Ivanov, I., Apostu, M. and Cristea, V. (2014). A multi-cloud platform for building cloud applications, *Proceedings of the International Workshop on Multi-cloud Applications and Federated Clouds*, pp. 9–16.

Nashold, K. and Krishnan, P. (2020). Using lstm and sarima models to forecast cluster cpu usage, *arXiv preprint arXiv:2007.08092* .

Patel, M., Singh, R. and Thakur, A. (2023). Hybrid time-series models for cpu and memory forecasting in cloud systems, *IEEE Transactions on Cloud Computing* .

Shrivastav, P., Agarwal, A. and Verma, R. (2023). Forecasting cloud workload trends using deep learning and statistical models, *International Journal of Computational Intelligence* **39**(2): 112–125.

Thiyagarajan, R. B. (2023). Multi-cloud disaster recovery using terraform and devops pipelines, *Proceedings of the International Conference on Cloud Engineering (IC2E)*, IEEE, pp. xx–yy.

Zhong, L., Sapozhnikov, A. and Margolis, A. (2019). Beyondprod: A new model for cloud-native security, *Journal of Physics: Conference Series*, Vol. 1237, IOP Publishing, p. 022033.