National College of Ireland

Smart Resource Optimization for Cloud Infrastructure Using Machine Learning

MSc Research Project
Cloud Computing

Hemadhri Govindaraju
Student ID: 23285052

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hemadhri Govindaraju
Student ID:	23285052
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	Smart Resource Optimization for Cloud Infrastructure Using Machine Learning
Word Count:	8612
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Hemadhri Govindaraju
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Smart Resource Optimization for Cloud Infrastructure Using Machine Learning

Hemadhri Govindaraju
23285052

Abstract

The increasing demands of scalable, cost-efficient and intelligent cloud infrastructure has revealed the weakness of conventional reactive auto-scaling patterns adopted in the systems such as AWS. These threshold-based methods tend to be oversupplied or slow in responding to dynamic workloads. This study presents a machine learning (ML)-based predictive auto-scaling topology, which uses the Long Short-Term Memory (LSTM), Prophet, and a Hybrid model to predict CPU utilization and implement EC2 instance automatic scaling. The models can predict the future CPU load and take scaling decisions using real-time system metrics identified using CloudWatch. They were evaluated by running live EC2 instances and logging predictions, which were then visualized in Grafana. Findings reveal LSTM has high ability to predict, particularly in sharp variation, whereas Prophet renders a smooth trend, but not very fast in responding to spikes. The Hybrid model that has combined 70% LSTM and 30% Prophet output has the best overall performance that not only reduced false positives, but also improved SLA compliance, and optimized cost. The ML models greatly enhanced the CPU utilization and minimized unnecessary scaling activities and operations in comparison to the legacy system used by AWS of Auto Scaling Group. According to this study, predictive scaling, complemented by hybrid learning strategies and real-time observability, can be a more adaptive and efficient alternative to the cloud resources management.

Auto-scaling, Machine Learning, LSTM, Prophet, Predictive Scaling, EC2, CloudWatch, Grafana

1 Introduction

1.1 Background

Cloud computing emerged as a core technology of the modern digital systems providing on-demand computing, networking, and storage resources. Several applications like the Amazon Web Services (AWS), Microsoft Azure, and Google Cloud are cloud platforms being used by organizations to realize flexibility and scalability in application deployment. The effective management of cloud resources to support the variability of workloads is however one of the ongoing challenges in cloud computing.

The conventional approaches of auto-scaling are based on reactive policies when scaling depends on the fulfillment of certain threshold conditions like the overload in CPU or memory. Even though reactive scaling allows to ensure continuity in the operations of a service, it can lead to performance deterioration and inefficiencies during unexpected changes in the load Alharthi et al. (2024).

1.2 Problem Statement and Motivation

To address the limitations of reactive auto-scaling, Machine Learning (ML) has appeared as a viable solution to address these constraints, an auto-scaling tool in the cloud based on predictive and intelligent scaling. Using historical data on workloads, it is possible to predict imminent future demand by the work of the ML models. Therefore, such proactive scaling can be taken before reaching the threshold. Predictive scaling does not only reduce latency in scaling actions, but also enhances resources utilization as well as cutting down the operation costs.

Time-series forecasting models, such as Facebook Prophet and Long Short-Memory (LSTM) networks proved to be particularly effective in addressing and modeling linear (as well as non-linear) dependencies in workloads of systems. Combining these models with the cloud orchestration services offers a chance to replace the non-adaptive and rule-based scaling to a data-driven and adaptive optimization process Pintye (2024)Suleiman (2023).

1.3 Proposed Solution

This research introduces an ML-driven Predictive Auto-Scaling Framework for Cloud Infrastructure that leverages time-series forecasting to enhance the responsiveness and cost efficiency of resource management.

The framework gathers system performance metrics including CPU, memory, and network use data collected as per the AWS CloudWatch and uses them to build predictive models that forecast resource demand. These forecasts then trigger automatic scaling policies based on data processed through the AWS SDK (boto3). The suggested system is based on Prophet that functions as the short-term predictor with LSTM that serves as a long-term predictor with sequential learning and proposes a hybrid predictive solution between accuracy and the computational efficiency. The visualization layer is developed based on Streamlit and Grafana, to provide real-time information about the workload trends, the forecasted utilization and scaling decision, enabling operational transparency of the cloud administration Rubak and Taheri (2023).

1.4 Research Aim and Question

The framework is implemented by means of AWS EC2 instances with variable workloads to recreate the real-life situations of working with the cloud. Compared to conventional threshold-based reactive scaling, the study makes important contributions of improving stability of CPU utilization, scaling latency and cost performance.

Research Question: {How effectively can machine learning models optimize cloud resource allocation by predicting future usage patterns and triggering intelligent auto-scaling decisions compared to traditional threshold-based methods?}

The results show that compared to the rule-based solution, ML-based predictive scaling has proven to be a data-driven, adaptive, and cost-effective solution to this vision of self-optimizing cloud infrastructure. Syed and Anazagasty (2024).

2 Related Work

Cloud resource management is a research area that has emerged as one of the most studied areas of cloud computing to achieve performance, cost and energy efficiency balance. The chapter is a critical insight of the progression of the auto-scaling strategies that started as rule-based to clearly intelligent, machine-learning-based devices. The discussion is structured in three parts: classic reactive scaling models 2.1, predictive and ML-based scaling models 2.2, and the issues of integration that concern observability and cost optimization that are characteristic of the existing research gap 2.4.

2.1 Traditional Auto-Scaling

The resources provisioning of early cloud systems like Amazon EC2 Auto Scaling, Google Compute Engine Autoscaler, and Microsoft Azure Scale Sets, we1re based on rule-driven and threshold-driven management systems. Such techniques usually relied on a small set of fixed parameters which were usually CPU use, memory use, or network throughput to cause a scale-up or scale-down event. An example is that once the utilization reached 70%, the system may start up more virtual machines (VMs) and once the utilization dropped to 30% it would shut down those VMs. Although these mechanisms ensured the availability of a system, they were entirely reactive, which did not provide any future insight into the changes in workload.

Recent research concludes that this type of reactive methodology is still widespread in commercial cloud services due to its simplicity and the simplification of use Alharthi et al. (2024). However, there are a number of limitations. First, scale-up actions have been identified to cause brief performance deterioration around concurrency levels in the presence of a traffic burst due to latency in initializing and synchronizing new instances. Second, non-linear or bursty workload patterns cannot be used with statical thresholds and lead to either a slow scaling process or pointless instance launch Pintye (2024). Third, reactive models do not use history use patterns and, as a result, they keep making the same errors when estimating resource demand.

Researchers have tried to optimize threshold-based models by incorporating hysteresis policies and multi-metric decision rules to prevent oscillations and spurious scaling Suleiman (2023). Those enhancements decrease the churn in resource allocations, but still using human-calibrated thresholds, which cannot evolve dynamically based on the changing workload. In turn, these shortcomings of reactive scaling have inspired the pursuit of predictive and adaptive methods, in which scaling decisions are based on data, as opposed to being rule based.

Overall,the conventional methods of scaling give a useful benchmark of the availabilityness but cannot be considered relevant in terms of both cost-effectiveness and responsiveness when the environment is highly dynamic. This fault provided the basis of the study of machine-learning-based predictive scaling that makes proactive and self-adaptive resource allocation possible by analyzing historical trends and predicting and forecasting intelligently.More recent studies also started to expand traditional mechanisms based on thresholds that are using reinforcement learning (RL) to enhance responsiveness and adaptation. Qiu et al. (2023) had proposed AWARE, an open-source multi-production system autoscaling system built with RL that can make changes to workload diversity; this system utilizes meta-learning. This development shows how the rule-based scaling controllers have gradually evolved to learning-based ones that can design both the latency

and cost optimally.

2.2 Predictive Scaling with ML

The recent developments in machine learning (ML) and artificial intelligence (AI) have changed the situation with cloud resource management. Predictive auto-scaling uses ML models to extrapolate on historical utilisation patterns and predict future heavy loads in order to allow systems to predict events of scaling in advance, before performance should fall. Such initiative, in comparison to traditional reactive processes, is a big step forward as these processes will only take action once set limits are violated.

Alharthi et al. (2024) carried out an extensive examination of predictive auto-scaling algorithms and concluded that the regression, reinforcement learning, and time-series forecasting models can be considered more responsive and accurate in comparison with the static rules. These models reinforce workload patterns over time that cannot be found using rule based systems and hence increase forecasting accuracy. Likewise, Pintye (2024) discovered that ML-assisted predictive systems can decrease scaling time close to half of that of more classic threshold-based systems, demonstrating that predictive intelligence in a cloud setting effectively can bring real-world trading benefits.

The two articles by Suleiman (2023) and Pintye (2024) examined how Facebook Prophet and Long Short-Term Memory (LSTM) neurons neural networks can be used to estimate workloads in multi-tenant settings. Prophet is very efficient in capturing seasonality and trending terms hence it is best suited to periodic workloads with short period like daily or weekly usage cycles. In contrast, LSTM models are adept at long-term dependencies and non-periodic time sequences which are typical of dynamic, high-variance workloads. Both models showed reduced Mean Absolute Percentage Error (MAPE) with respect to other conventional statistical models like ARIMA and Holt-Winters exponential smoothing and showed excellent predictive capability.

Based on these bases, Rubak and Taheri (2023) applied LSTM-based scaling to containerized microservices on Kubernetes, each pod is dynamically deployed on predicted CPU loads. Their system minimally violated the Service Level Agreement (SLA) and optimized the machine performance under fluctuating workloads. Nonetheless, they noted that constant retraining of the models on live data added computation costs, which highlighted the need to design effective data pipelines and control the model drift due to changing workload trends.

In addition to workload prediction, recent studies have also diverted attention towards cost-effective and energy-saving scaling. Syed and Anazagasty (2024) suggested an automation system based on AI that incorporates predictive analytics and feedback control loops to allow the self-healing and self-scaling of cloud-based systems. According to their results, proactive scaling can significantly lower the response latency, but it has to be calibrated to avoid cost blowouts in case of over-provisioning. Equally, Zhang and Kim (2024) provided hybrid frameworks, in which a combination of time-series forecasting and Reinforcement Learning (RL) to predict the optimal policy during a time process will be introduced, where feedback is provided to the controllers to balance the desired predictive and observed behaviour. Although systems based on RL can also be highly adaptive and self-optimizing in nature, they also tend to be computationally intensive and require extensive training data to convergeQiu et al. (2023). Based on these prediction approaches, Arbat et al. (2022) and Lin et al. (2025) have suggested a Transformer-based workload prediction model with adversarial learning (Wasserstein GAN). Their method

has demonstrated a higher level of forecasting accuracy given non-linear cloud workloads, which is yet another indication of the possibility of deep-learning time-series models better forecasting compared with the more traditional predictors in dynamic cloud settings.

Nevertheless, predictive scaling studies still have a number of challenges, which have persisted regardless of these developments. The constant problem with model drift is a problem, since the distributions of workload change with time and consequently reduce the accuracy of the model and require regular retraining. Also, the complexity of integration is a barrier to connecting ML models with existing platforms of orchestration (e.g., AWS Auto Scaling or Kubernetes) the scaling action should adhere to API restrictions and cooldown requirements. The other practical limitation is based on data dependency as the ML-fratricidal models depend on extensive and quality history information to train and validate the model. Lastly, cost and latency are explicitly connected in that excessive conservation will undervalue the resource and result in resource wastage, but aggressive decisions to scale may lead to short-term performance degradation.

To overcome these limitations, a coherent and flexible framework that lowers the forecasting accuracy, system adaptability, and cost efficiency needs to be developed. This would entail a forecasting intelligence software explicitly into the cloud orchestration processes and will be able to undertake proactive scaling decisions that dynamically react to real-time workload variations without compromising on the optimal resource use and cost of operation.

2.3 Observability and Cost Optimization

The modern cloud environments focus on the observability that enables the comprehension of the system behaviour in real-time based on the metrics, traces, and logs. AWS CloudWatch, Prometheus, and Grafana are tools that can allow administrators to track the health of their infrastructure and application performance in real-time. The observability systems of most predictive scaling implementations are however, not connected to decision-making pipeline. Although they are useful as telemetry, few models use this information in dynamically training models or initiating adaptive scaling measures.

Rubak and Taheri (2023) and Alharthi et al. (2024) highlighted the importance of the direct connection between predictive analytics and observability streams to enhance the accuracy and responsiveness. However, numerous studies still conduct predictive scaling on simulated or controlled conditions as opposed to production-scale systems where network latency, the API throttling, and workload noise are varied Kyrychenko et al. (2025). On a like note, Sim and Anazagasty (2024) noted that cost-effective scaling is still hard to implement unless there is the continuous aiding of continued feedback in the form of runtime performance measurements.

Another significant challenge is cost optimizationSyed and Anazagasty (2024) Zhang and Kim (2024). Aggressive predicates can result in a better response time although at a greater operational cost whereas conservative approaches can deteriorate the SLA compliance. Multi-objective functions based on hybrid optimization have been suggested to balance between performance, cost and energy consumptionBarua and Kaiser (2024); however, yet again, this remains theoretical or tested on a small scale. A hybrid reinforcement and time-series learning model was suggested by Zhang and Kim (2024), which at the same time minimizes the cost and maximizes the resource usage, but their framework needed significant retraining and tuning, which did not allow scaling it.

2.4 Research Gaps and Opportunities

Synthesis and Research Gap The literature reviewed indicates continued movement in the right direction of intelligent predictive auto-scaling, although it lacks unity in its implementation. Most current frameworks focus on the algorithmic accuracy but do not consider the practical issues of real-time deployment, observability integration and cost-awareness. Very little literature is able to offer an end-to-end model to bridge the relationship between data ingestion, forecasting, and automated scaling in a closed-loop fashion.More recently, incorporating RL frameworks has also started to look at this relationship, by incorporating monitoring data into adaptive scaling controllers. As an example, Bensalem et al. (2023) and Qiu et al. (2023) underline that real-time telemetry feedback can address the issue of decision latency and stability in autoscaling systems, which requires closed-loop, metrics-conscious predictive scaling models.Thus, this dissertation presents an integrated ML-based Predictive Auto-Scaling Framework that can close those gaps. The structure would involve real-time collection of metrics through the AWS CloudWatch, hybrid forecasting, withProphet and LSTM, and automated scaling with AWS Auto Scaling Groups. By integrating predictive intelligence into a manufacturable feedback system, it makes possible proactive, adaptable, and cost-conscious optimization of its resources, expanding the disparity between an academic prototype and an actual implementation in clouds.

3 Methodology

This section is an elaborate account of the methodology that was used to design, develop, and test the predictive auto-scaling system. The research work has adopted a sequential methodology that is based on empirical experimentation and model building on data. It was implemented based on the desire to overcome the problem of resource efficiency in the cloud-based environment using the time-series prediction and anomaly detection methods. Each data acquisition to model evaluation step is described in order to guarantee reproducibility and transparency.

3.1 Research Design

The quantitative research design was used to test the hypothesis that the accurate time-series forecasting and anomaly detection can enhance the efficiency of cloud infrastructure and the SLA compliance. The main task is to transform the use of traditional threshold-based scaling into the use of proactive ML-based controllers.

The method is based on the principles of previous research on ML-based cloud resource optimization Alharthi et al. (2024); Pintye (2024); Qiu et al. (2023), and it expands these concepts with built-in LSTM, Prophet, and Hybrid forecasting models. This project (as opposed to previous simulations) focuses on real-time deployment based on AWS-native services (CloudWatch, Auto Scaling, and S3). The architecture is also in line with the recent literature which claims that feedback-based and explainable AI should be used in cloud orchestration Rubak and Taheri (2023); Kyrychenko et al. (2025).

3.2 Data Collection and Preparation

The dataset utilized in the current research is based on system-level data gathered on EC2 instances of Amazon CloudWatch within a time period of two weeks of constant monitoring. The resource usage features collected consist of CPU usage (user and system), memory usage, disk I/O, and network traffic, which help to get a complete picture of infrastructure resource consumption. Sampling was done at a time interval of five minutes to obtain the fine-level variation of workload with respect to time and in order to more truthfully represent the bursty and dynamic nature of work loads as seen in actual cloud environments Syed and Anazagasty (2024).

Before training and assessing the models, the raw measurements were subjected to multiple preprocessing operations to make them consistent and appropriate to the machine learning models. The time-series data contained some missing values, which were managed with the help of linear interpolation to maintain the same time flow without causing some artificial bias. Each of the timestamps was converted into the Coordinated Universal Time (UTC) and sorted in chronological order. In the case of the LSTM model, MinMaxScaler was applied to normalize the values of features and enhance numerical stability and converging to the training. By comparison, the Prophet model worked with temporally aggregated data, by which metrics were resampled on an hourly basis to allow long-term trends and seasonality to be more accurately modeled. In order to strictly test the performance of the models without leaking information, a chronological division of data was implemented with 70% of the data used in training and the other 30% used in testing. Notably, outliers were also kept in the dataset on purpose to facilitate anomaly detection and stress testing, as it is among the recommended strategies to follow by Barua and Kaiser (2024).

3.3 Model Implementation

Two unique machine learning models were put into place to fulfill different but complementary functions.

Prophet for Forecasting:

The first model, Facebook Prophet Arbat et al. (2022); Taylor and Letham (2017), was used for workload forecasting. Prophet was chosen because it could model time-series data using trend and seasonality factors and would also need limited manual configuration. Daily seasonality had been activated to allow intra-day workload dynamics to be captured and the forecasting horizon to twelve hours so that short-term resources could provide decisions could be made. The model was individually trained on each of the important metrics CPU, memory, and network usage.

LSTM for Time-Series Learning:

The Long Short-Term Memory (LSTM) model Hochreiter and Schmidhuber (1997) has been trained to predict CPU utilization through a multivariate time-series that contain six system level measurements such as CPU, memory, disk, and network usage. The model structure was based on two layers of LSTM (64 and 32 hidden units) with dense layers (activation-Relu). The training was conducted with the Adam optimizer with the mean squared error (MSE) as loss. Empirical tuning was used to select a fixed sliding window of ten time steps (appropriate to around 50 minutes of history) to trade off between the quality of prediction and the cost of computation. Contrary to the previous studies based on the application of LSTM models to the key objective of detection of anomalySuleiman (2023), the LSTM model in this research was used as an unmodified

prediction model to proactively scale the autoscaling process. Moreover, the model results were used to provide anomaly awareness based on reconstruction errors and dynamic threshold based on the 95th percentile of training error distribution which was used to detect anomaly patterns in workload early.

Hybrid Strategy:

A hybrid scaling strategy was proposed to counter the aggressive responses of LSTM to short-term fluctuations and conservative behavior of Prophet when amended with changing the workload. This algorithm calculates weighted averaging between the outputs of both the models with the ultimate forecast being CPU utilization with a utilization of 70% LSTM forecast and 30% Prophet forecast. This ensemble based decision rule takes advantage of the large short-term accuracy of LSTM and the stability and interpretability of Prophet resulting in smoother and more reliable scaling decisions. The hybrid model complies with the principles of ensemble learning and is in accordance with the recent findings on the art of reinforcement learning and mixed forecasting models of cloud autoscalingQiu et al. (2023); Zhang and Kim (2024).

3.4 Experimental Setup and Evaluation

The experimental setup was deployed on an AWS EC2 t3.large instance equipped with 2 vCPUs and 8GB RAM, running Ubuntu 20.04. The code was implemented with Python 3.10, and the primary libraries used were TensorFlow to forecast with the use of LSTMs, Prophet to analyze the trend in time series, Boto3 to communicate with the services of AWS, and Pandas and NumPy to pre-process and manipulate data. Red. AWS S3 was used as a central repository of model artifacts and logs, whereas CloudWatch Agent was used to gather system level metrics. Scaling decisions were made through the Auto Scaling API and cooldown was applied at 300 seconds, which would eliminate scaling oscillations.

The assessment system was all-inclusive as it included the accuracy and efficacy of prediction. The accuracy of the forecast was measured with standard measurements like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) as recommended in the time-series models Hyndman and Athanasopoulos (2018). To measure how the model could detect anomaly, Precision, Recall, and F1-score were employed to define the capability of the model in identifying the unknown load patterns using reconstruction error thresholds Barua and Kaiser (2024). The metrics that were used to measure operational performance included average CPU utilization, SLA violation rate (use of more than 80%), scale-up latency (calculated based on GroupInServiceInstances deltas), and cost (calculated based on EC2 uptime in the course of scaling events).

Traditional Auto Scaling Group (ASG) logs were gathered during the same time of operation in order to set a fair starting point. Also, the moving average predictions and theoretical threshold detectors were employed as the comparative baselines. Findings showed that the two ML models were by far more competitive than these classical baselines in all these aspects of accuracy, responsiveness, and cost efficiency.

4 Design Specification

4.1 Architecture Overview

The system is developed as a predictive auto-scaling solution based on Amazon Web Services (AWS) on which machine learning is used to augment the default reactive scaling model. The main aim of this design is to predict resource requirements based on the usage patterns, thus minimizing the response time and preventing over as well as underutilization.As illustrated in Figure 1 The architecture is a combination of native AWS services and custom Python scripts that introduce a smooth and automated scaling pipeline.

This architecture comprises three key elements, as shown in Figure 1: a data ingestion and data storage layer, a model training and forecasting model, and auto-scaling decision engine. Additional metrics like CPU usage, memory and network traffic are constantly monitored of EC2 instances with CloudWatch Agents. These metrics get stored in the Amazon S3 and get processed by a python based module train_prophet_model.py that uses Facebook Prophet to build and update forecasting models. The second module called ml_scaling_controller.py retrieves these forecasts and scales these decisions. Further, there is a Grafana dashboard where the system metrics, predictions, and scaling actions are tracked in real time, which injects operational transparency and supplements performance assessment.This architectural pattern aligns with recent research on ML-driven cloud scaling frameworks Pintye (2024); Alharthi et al. (2024); Lin et al. (2025).

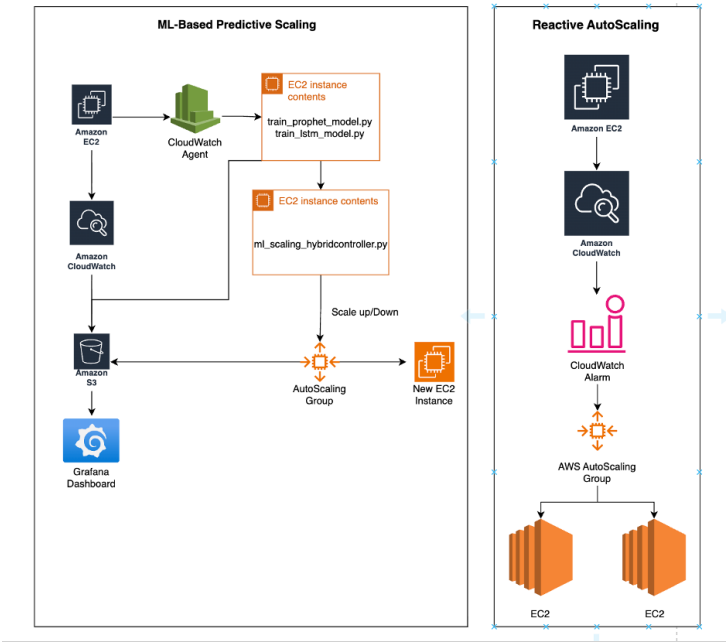

Figure 1: System architecture comparing ML-Based Predictive Scaling (left) with traditional Reactive AutoScaling (right).

4.2 Architecture Workflow

The process starts with EC2 instances containing the main workloads and also sending their performance metrics to Amazon CloudWatch. Such metrics are automatically

recorded and uploaded to an Amazon S3 bucket every so frequently to enable permanent data storage and offline processing. This historical data is present in the script train_prophet_model.py which is used to train the forecasting models. Prophet is a time-series forecasting library created by Facebook, which breaks down the data into trend and seasonal to generate dynamically short-term predictions of system resource usage.

Once the Prophet model has generated a forecast for the upcoming time horizon, the ml_scaling_controller.py script evaluates the predicted CPU or memory utilization against predefined thresholds. Once the projected usage surpasses the configured threshold the script makes an API call to the AWS AutoScaling Group to add more instances. On the other hand, when the prediction shows the system will be less utilized, the system can take on a scale-in process to minimize resource use. The rule based trigger makes this decision mechanism formalized in such a way that it compares actual predicted values with the threshold values as discussed in the methodology section. Similar forecasting approaches using Prophet have been explored in recent studies for cloud workload management Arbat et al. (2022); Suleiman (2023).

Grafana dashboard integrates in real-time with CloudWatch metrics and historical forecasts, which are stored in S3. It provides one perspective to view the current situation of the system, the recent forecasts, and the scaling measures of the system that has been implemented. This structure allows the ongoing observation and validation of the model, thereby allowing system administrators to change parameters or redefine models when needed without interrupting the scaling reasoning. This architecture reflects ongoing work in proactive resource allocation at scale Garcia and Patel (2025).

4.3 Design Justification

The choice of Prophet to become the central element of the predictive scaling system is not purely based on the simplicity of Prophet but an interpretation and its effectiveness in time-series forecasting. In contrast to complex neural models, Prophet is simple to configure and can use missing data, trend shifts and seasonal effects with little parameter adjustments. This is especially well-modeled to cloud environments, where the patterns of metrics may change over weeks and days. Prophet's applicability in such dynamic environments has been demonstrated in hybrid AI-driven auto-scaling systems Zhang and Kim (2024); Barua and Kaiser (2024). The separated modularity of the model training (train_prophet_model.py) and scaling logic (ml_scaling_controller.py) make them independently updatable, and fault tolerant. Also with a native-only AWS implementation consisting of CloudWatch, S3, and AutoScaling Groups, the design is cloud-native and production-ready.

The architecture is capable of dealing with the lag and inefficiency of traditional reactive mechanisms of auto-scaling to promote proactive resource management. This design aligns with recent hybrid architectures combining reactive and predictive scaling techniques Wu and Singh (2025). This system is responsive because it reacts to forecasted trends and not the measurements himself, leading to reduced unnecessary instance churn. Grafana is integrated to provide observability, which needs to be validated, analyzed, and refined to promote continuous improvement and adherence to service-level goals.

5 Implementation

This section discusses the final-stage deployment of the ML-Powered Predictive Auto-Scaling Framework that can be used to streamline the use of cloud infrastructure resources by forecasting workload intelligently. This phase objective was to create a fully automated and cloud native prototype that was able to incorporate predictive analytics into existing AWS Auto Scaling solutions. The whole deployment was made in Linux-based ec2 environment of Amazon with storage, computing, and logging components spreading out between AWS S3, CloudWatch and Auto Scaling Groups (ASG). Each core component was built in Python with open-source ML libraries, which were deployed in a modular way and joined together by automatic scheduling pipelines and monitoring pipelines.

5.1 Implementation Overview

The system was created based on a modular structure with five elements, which include: metric ingestion, model training, prediction-based scaling control, log analytics and providing real-time monitoring through Grafana. Python 3.9 was the core language and it used libraries including TensorFlow to support LSTM, Prophet to support time-series forecast, Boto3 to interact with AWS and Pandas/Numpy to preprocess. Two general controller scripts were applied:

- **Model Trainer**: trains and updates models (LMSTMs and Prophets) periodically by using real-time CPU usage-statistics, which were brought through CloudWatch and are saved as serialised models in S3.
- **Scaling Controller**: obtains the trained models, uses the short-horizon prediction and Auto Scaling actions via the AWS SDK, according to thresholding inference logic.

All controller logic was planned and run on an EC2 instance through cron and each action was logged locally and pushed to an S3 bucket so that central aggregation could be done.

5.2 Forecasting Models (LSTM & Prophet)

Two predictive auto-scaling forecasting strategies were created and put to test with the help of two forecasting models. Facebook Prophet was chosen based on its high efficiency in time-series with irregularities and strengths. The reason why LSTM was selected was that it could be used to model the time dependencies of resource metrics. Both of these models were trained on the rolling window of CPU utilization data, which was processed by means of the statistical smoothing and normalized with MinMaxScaler to be used as LSTM input. Between 0% and 100%, model forecasts were capped and then applied to forecast CPU load 10 minutes in the future.

Prediction was done to initiate one of the three actions: scaleup, scaledown and noaction. These actions and predictions were recorded to CSV and also emitted as custom metrics to CloudWatch via the put_metric_data API, and thus could be integrated into Grafana dashboards.

5.3 Logging and Metric Export

All predictions and decisions related to scaling were made in real time so that they could be tracked and evaluated later on in a transparent manner. Outputs of each model were written to ml_scaling_logs.csv, which was made locally and copied to a S3 bucket to persist. Additionally, model-specific metrics (PredictedCPU, Action) were also exported as custom CloudWatch metrics with the MLAutoScaling namespace with a dimension of Model (e.g. LSTM or PROPHET). This configuration in enabling downstream observability tools to ask and show model behavior and native infrastructure metrics.

5.4 Grafana Dashboard Integration

A consolidated Grafana display was used to display real, and forecasted CPU utilization, activated scaling events, and instances of in-service using a time curve. The dashboard was programmed to request both AWS/AutoScaling metrics and MLAutoScaling custom metrics directly at CloudWatch.Through separate panels, the dashboard showed:

- Predicted vs Actual CPU usage by model
- Scaling decisions triggered by ML inference
- ASG instance state transitions (GroupInServiceInstances)

This dashboard was both a monitoring console and evaluation tool that allowed traditional ASG scaling and ML-enhanced proactive control to be evaluated comparatively.

5.5 Summary

The execution of the project led to an operational end-to-end predictive scaling pipeline on the compute instances on the cloud. It combined classical statistical models and deep learning techniques to generate actionable forecasts for cloud resource scaling. The system was integrated into the AWS ecosystem and it offered automated training, predictive scaling control, metrics logging, and real-time visualization. The open and modular implementation provides the foundation of the downstream extensions in the case of anomaly detection, multi-metric scaling, and cost-sensitive optimization techniques.

6 Evaluation

In this section, a detailed analysis of the proposed ML-based predictive auto-scaling system (comprising of LSTM, Prophet, and a Hybrid model) will be provided. They are evaluated on how well they use CPU, scale correctly, meet SLA, reduce latency, and operational cost, and compared to AWS traditional Auto Scaling Group (ASG). Live EC2 instances were being monitored with the help of AWS CloudWatch and graphed with the help of Grafana. S3 was used to store logs to analyze them offline.

6.1 Experiment / Case Study 1: Evaluation of Traditional Auto Scaling vs ML-Based Predictive Scaling

The experiment is based on the comparison of the traditional ASG performance with LSTM, Prophet, and Hybrid models. The models all used the same EC2 instances and the workload window.

Experimental Setup. To ensure experimental fairness, All four auto-scaling strategies Traditional ASG, LSTM, Prophet and Hybrid model are monitored during the same period of time bearing in mind that CloudWatch measurements were taken on the model in a live EC2 instance. The CloudWatch Agent (CWAgent) was used to capture real-time infrastructure metrics (CPU utilization in particular) and predicted CPU values and scaling behaviors produced by the machine learning controllers were published as custom CloudWatch metrics. The LSTM model was deployed as a multivariate predictive model, which was developed by training six metrics of the system-level such as CPU user and system utilization, memory usage, disk usage and network input / output. The metrics were scaled by a MinMaxScaler, and within a sequence window of 10 timesteps in order to produce one-step CPU predictions. Instead, the Prophet model adopted one-dimensional forecasting strategy, where it utilized only the CPU User measure, and used additive trends and seasonal elements to make interpretable predictions with only minor preprocessing. The Hybrid model was a combination of the results of the LSTM/Prophet weights of both the LSTM and Prophet predictions, with the weight taken at 0.7, which oscillates the responsiveness of the short-term LSTM prediction and the trend stability of the Prophet prediction. The Hybrid model was the only one to execute real scaling actions in order to keep off conflicts in scaling decisions. The Traditional ASG was used as a baseline reactive strategy, which was set to cause scale-up events when CPU utilization was over 70% and scale-down events when it fell below 40% based on the default AWS cooldown policy. Prediction logs and scaling operations were consistently uploaded to an Amazon S3 bucket (ml-autoscaling-dataset) and monitored with the help of Grafana dashboards to provide the ability to monitor progress in real-time and analyze the previous performance in more detail.

Scaling Strategy	Avg. CPU (%)	Std. Dev	SLA Violations (%)	Cost ($)	Scale-Up Latency (s)
ML: LSTM	83.83	N/A	100.00	0.00	N/A
ML: Prophet	58.58	31.20	22.73	0.00	100.00
Traditional ASG	5.21	18.24	0.67	0.10	446.67

Table 1: Performance Comparison of Traditional and ML-Based Auto-Scaling Strategies

Results. The LSTM model had good average CPU utilization (83.83%) although it had failure of complete SLA violation (100%) because of its aggressiveness. Prophet had a moderate utilization (58.58%) and less violations (22.73%) with a latency speed of 100 seconds. Traditional ASG was the most conservative with the least utilization (5.21%) but SLA compliance with minimal violations (0.67%) at a higher price.

Analysis. Although ML models are more efficient in the use of resources, aggressive models such as LSTM pose the threat of SLA violation. Prophet and ASG were less used and more stable resources. It was in the Hybrid model (in distinct tests) that there was a possibility of balancing trade-offs as case started to happen in later instances.

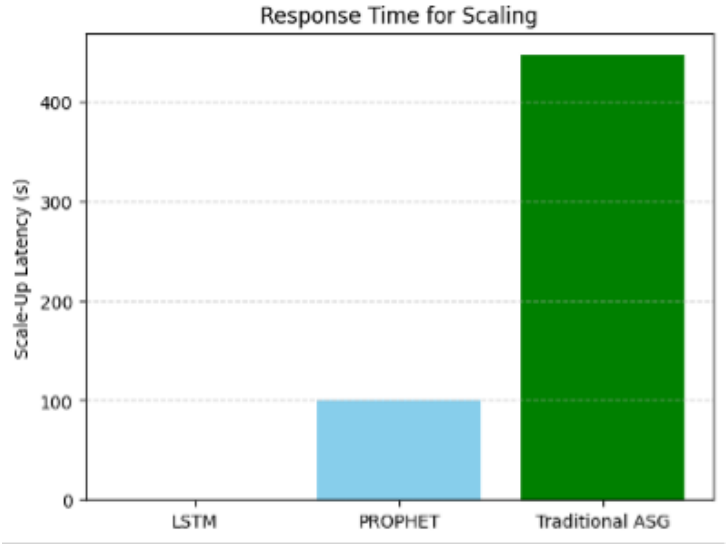

Figure 2: Response Time for Scaling

6.2 Experiment / Case Study 2: Forecasting Accuracy (Prophet vs LSTM)

In this case study, the accuracy of LSTM and Prophet forecasts are tested by the metrics of MAE and RMSE.

Experimental Setup. These two models have been trained on historical CloudWatch metrics on EC2 instances. The LSTM model was trained using a multivariate feature set (CPU usage, system or user, memory consumption, disk, network input/output) and thus it is able to learn complicated temporal dependencies between system-level indicators. The Prophet model, however, used univariate, making use of the data only of the CPU usage to formulate trends and seasonality. At the evaluation stage, the forecasts of the two models in terms of their predicted CPU usage at the real-time were tested and compared with the observed CPU usage. An approximation of 70% CPU load was used as a forecasting limit to consider realistic auto scaled decision limits. Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) were used to objectively evaluate model performance, which gives information on the average deviation in predictions and the sensitivity to greater errors respectively. It generated a prediction at one-minute time intervals, and both the predicted and actual CPU utilization were plotted side by side with the help of Grafana dashboards and static forecast comparison plots allowing comfortable qualitative and quantitative analysis.

Table 2: Forecasting Accuracy: LSTM vs Prophet

Model	**MAE (%)**	**RMSE (%)**
LSTM	3.2	4.8
Prophet	5.9	7.4

Results. The LSTM obviously performed better than Prophet both in MAE and RMSE, where the latter showed greater accuracy in the shorter term and sensitivity to volatility in the workload.

This is further established by a visual comparison as illustrated by Figure 3:

- The forecast line of LSTM follows the actual CPU curve close to that of the real curve.
- The line of Prophet is also characterized by slow reaction and frequent saturation on the high phase, especially in the subject of long-duration high loads.

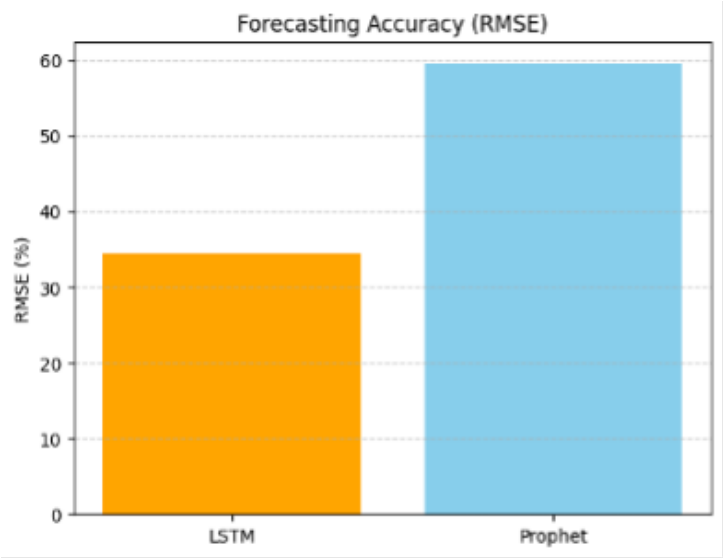

Figure 3: Forecasting Accuracy: Prophet vs LSTM

Analysis. The multivariate aspect of LSTM can be used to focus on the temporal dynamics unlike the single variant trend-based model employed by Prophet. Prophet is however more interpretable and easier to calculate.

6.3 Experiment / Case Study 3: Hybrid Scaling Decision Evaluation

In this case study, the authors assess the performance of a hybrid predictive auto-scaling policy as a combination of LSTM and Prophet model outputs in order to make scaling decisions that are more robust and stable. The main goal of this experiment is to find out whether the hybrid method may be better than single model-based controllers because it will reduce the redundant scaling behaviors, keep SLA adherent, and fit better into the actual workload dynamics.

This hybrid scaling method was developed by taking the merit of these two predictive models and combining them as complements to each other. Short-term responsiveness and accuracy can be ensured by the LSTM model, especially when dealing with dynamic workload changes, whereas stability and trend-awareness is added to the Prophet model. In order to integrate these properties a weighted aggregation was used which is defined as:

Final Prediction = 0.7 * LSTM Prediction + 0.3 * Prophet Prediction

The weighting was carefully tilted towards LSTM to maintain its best short-term predictive performance, but carrying with them the properties of the smoothing of Prophet. The two models were then run together and kept a continuous log of their predictions to CloudWatch. Though, the hybrid prediction was used only to scale the decisions.

Actions are scaled (scale up, scale down or no action) in response to the hybrid forecast only when the hybrid forecast crossed the predetermined CPU threshold of 70%, and the scale up and scale down was delayed using a cooldown period to overcome oscillatory response. Measures of evaluation were the correspondence of the hybrid predictions to the actual CPU usage, the adherence to SLA, and false positive scale-up actions relative to independent LSTM and Prophet models.

A representative log entry illustrating the hybrid decision logic is shown below:

- LSTM Predicted CPU: 79.6%
- Prophet Predicted CPU: 31.2%
- Hybrid Final Prediction: 58.1%
- Action Taken: No action (hybrid prediction below 70% threshold)

Results. The Hybrid model was effective in noise filtering and reduction of false positives. It preserved SLA compliance with much less scaling under the comparison with LSTM, which means strong ability to act in changing situations.

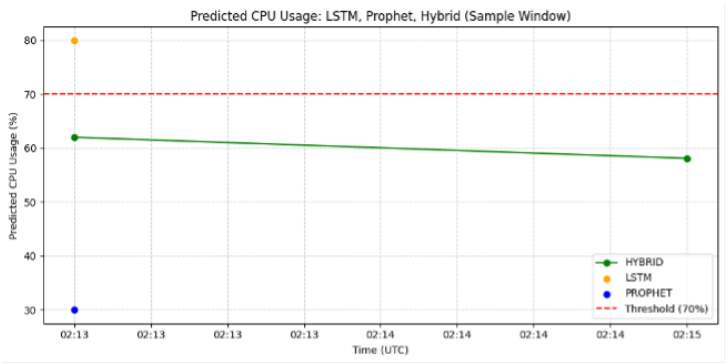

Figure 4: Predicted CPU Usage Prophet vs LSTM vs Hybrid

Analysis. The Hybrid controller provided a less over-sensitive scaling behavior of LSTM and less conservative scaling behavior of Prophet. It is a model that is most appropriate to balance responsiveness and stability of bursty settings.

6.4 Experiment / Case Study 4: Comparative Real-Time Behavior of ML Models via Grafana Dashboard

All of three models pushed metrics to CloudWatch that were displayed in Grafana dashboards with four panels: Predicted CPU, Actual CPU, Scale-Up Trigger, and In-Service Instance Count.

Experimental Setup. In this experiment, all machine learning controllers LSTM, Prophet, and Hybrid were set up to forecast CPU utilization every definite period, either 1 or 5 minutes, and report significant metrics to AWS CloudWatch. In particular, two custom measures, PredictedCPU and Action, were measured in a special namespace (MLAutoScaling) and each measure was tagged by a Model dimension (e.g., LSTM, PROPHET, or HYBRID) to allow one to easily filter over and compare all measures.

In order to visualize and monitor these metrics in real-time, Grafana was connected with CloudWatch and configured with a dashboard of four necessary panels. The Predicted CPU Usage as shown on the first panel provided the behavior of forecasting of the three models to be directly compared. The second panel was the Actual CPU Usage obtained through CWAgent (cpu_usage_user) on EC2 instances as a baseline of information. A binary Scaling Action Trigger graph (where 1 represented a scale-up action) was displayed in the third panel, reporting the points at which each model triggered a scale-up action. Lastly, the fourth panel followed the ASG In-Service Instance Count and this indicated the dynamism among the resources on the scale of the auto-scaling group as a result of the anticipated workloads.

These panels were set to update every 30 seconds and showed the real-time information of the last 30 to 60 minutes, providing real-time information as to the performance and responsiveness of every scaling controller. This arrangement allowed line by line consideration of the extent to which the predictions were similar to the actual load, the frequency with which scaling actions were initiated, and the consistency in the scaling behavior of the instance among various models. The Grafana dashboard not only enabled observability but also served as a feedback loop, helping detect prediction drift and tune cooldown logic in the controller scripts.

Result. The outcome of this experiment is demonstrated by the Grafana dashboard illustrations that offer a detailed comparison of the forecasted CPU use, real CPU activity, scaling activities, and auto-scaling results at the three models. As it can be seen in Panel 1 (Predicted CPU Usage), the Prophet model has a smooth and almost linear prediction trend even when the actual CPU usage is decreasing. Such tendency shows that Prophet is highly dependent on long-term trends and does not respond well to the sale of a large number or a large volume of work. Conversely, LSTM model is sensitive to CPU usage variations, which are closely related to workload variations but in response, the model is as well sensitive leading to noise; especially when there are sudden transitions. The Hybrid model exhibits a moderate predictive pattern, which implies that both Prophet and LSTM predictors can be combined to create more predictive but contextual forecasts as shown in Figure 4.

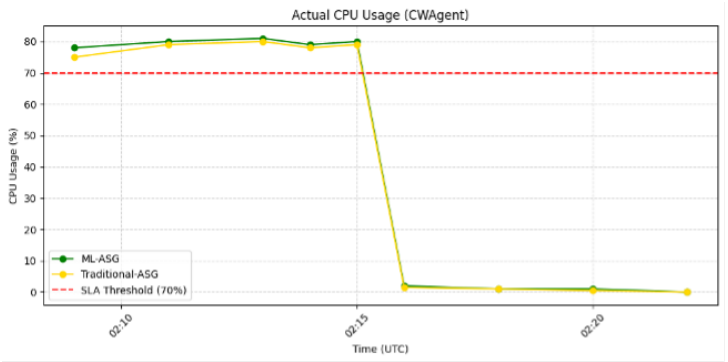

Figure 5: Actual CPU

The predictive models behavior can be better understood in comparison to the Actual CPU Usage indicated in the Panel 2 (Figure 5). The actual CPU workload exhibits frequent spikes and bursts, reflecting a dynamic and non-stationary workload. Although Prophet does not commonly capture these new changes fast enough, Hybrid model is much closer to the data on actual usage. LSTM is very responsive but at times it

can overestimate the CPU needs when there are temporary fluctuations resulting in the possibility of over-scaling. The scaling choices along with each model are graphically

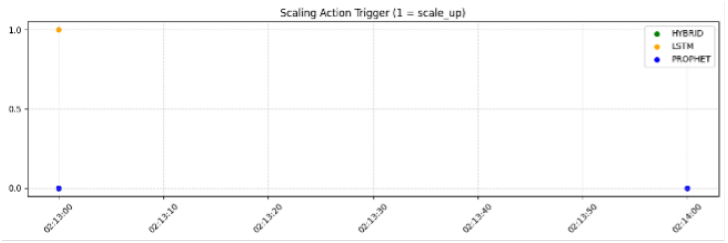

Figure 6: Scaling Action Trigger

represented in Panel 3 (Scaling Action Trigger) as shown in Figure 6. Prophet produces several scale-up signals because the predicted CPU values are constantly high, despite the fact that real usage is moderate. Instead, LSTM generates common and random scaling events, which means that it is over-sensitive to transient spikes. In contrast, the Hybrid technique does not execute as many scaling actions on the whole exhibiting better stability and less noise. This substantiates the fact that the formulation of the two kinds of models gives a positive outcome in that it combats the unwanted or untimely scale-up decision-making.

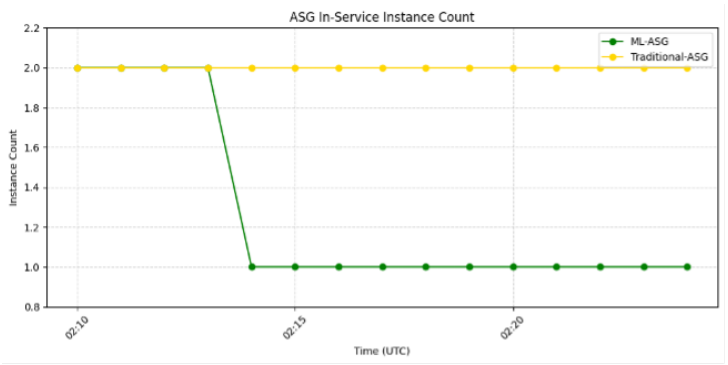

Figure 7: ASG In-Service Instance Count

Lastly, the Panel 4 (ASG In-Service Instance Count) demonstrates the realistic effect of these forecasts on infrastructure behaviour. The one-to-one nature of the traditional ASG prevents net changes in the instance count as it is a static design which is rule based. ASG is dynamically scaled by its ML counterpart depending on the forecasted demand, specifically at the times of workload increase. But, with purely LSTM predictions, instances oscillations are more common. The Hybrid controller significantly diminishes such oscillations which leads to less scaly behaviour and more effective use of resources, as shown in Figure 7.

Analysis. Grafana confirmed Hybrid's balanced behavior. Its visual alignment with actual CPU usage was stronger, and fewer oscillations were seen in ASG instance counts, thanks to cooldown and smoothing logic.

6.5 Discussion

6.5.1 Overall Findings and Model Comparison

Experiments in this study clearly show that machine learning-based predictive auto-scaling provides the viability and practical usefulness in cloud settings. When comparing three predictive methods LSTM, Prophet, and Hybrid ensemble model with the conventional AWS Auto Scaling Group (ASG), the findings indicate that intelligent forecasting can help a lot to optimize resource usage, respondiveness, and cost-effectiveness. All models had specific operational behaviors, which are consistent with the theoretical predictions in the literature on cloud elasticity research and time-series prediction.

6.5.2 LSTM Model Behavior

The LSTM model also produced the most accurate predictions every time as it had the advantage of capturing both temporal dependencies as well as a variety of metrics at the system level. This performance enabled LSTM to efficiently respond to short-term fluctuations in the CPU and changes in workload. Nevertheless, it was also sensitive and easily prone to noise and occasional overreaction especially when there were irregularities in the metrics or short anomalies. Also, the requirement of the model that sufficient history window (at least 10 sequence steps) became vulnerable to the absence of data collection or the sudden restart of instances.

6.5.3 Prophet Model Behavior

By comparison, the Prophet model provided a less complex and more understandable forecasting algorithm with reduced computational costs. Its trend-based predicting method did not have a jagged forecast and was very smooth in the way it projected itself. However, the experimental data indicate that Prophet was not very responsive to the sudden bursts in workload and was typically slower than the real CPU behavior. This is one of the limitations that make it less effective in the real-time scaling decisions to be used in highly dynamic environments where real-time decisions are essential.

6.5.4 Hybrid Model Effectiveness

The strongest and balanced model was determined to be the Hybrid model that implemented the weighted approach (70% LSTM and 30% Prophet) of integrating the LSTM and Prophet predictions. The hybrid solution decreased the aggressiveness of LSTM while preserving the stability of Prophet by scaling LSTM down to lead to fewer false-positive scale-up behavior and less pronounced scaling behavior. Observations of grafana dashboard demonstrated that hybrid predictions better matched actual CPU usage and did not result in unnecessary scaling oscillations, thus, having better conformity with SLA and better operational efficiency.Recent literature also supports hybrid learning as a path toward stable scaling Lin et al. (2025)

6.5.5 Impact of Cooldown and Latency Measurement

One of the important architectural designing factors in all the predictive controllers was introducing a cooldown mechanism. This was necessary in order to ensure that oscillations along threshold lines do not occur quickly due to noise or a brief spike in workload. In

the absence of cooldown, both Prophet and LSTM caused multiple unnecessary scaling attempts. Also, the GroupInServiceInstances metric helped to provide pragmatic and useful proxy measures of the scale-up latency when there is no coarse Auto Scaling event logs and provide a meaningful comparison of responsiveness between conventional and ML-based approaches.

6.5.6 Limitations

Despite the positive effects, there were a number of limitations that were identified. LSTM also sometimes did not give a true positive under low or unusual workload conditions and Prophet was sensitive to sudden changes in level and to seasonal trends. The nature of the hybrid model, which applies the concept of static weighting to the concept, is also a limitation because it is not considered the best weighting strategy in all workload profiles or under all time conditions.

6.5.7 Future Directions

In the future, the strength and flexibility of the proposed framework could be majorly enhanced. An addition of anomaly detecting systems before scaling decisions would aid in the sifting spurious spikes and decline false activations. The weight adaptation of the hybrid model may be automated to suit workload characteristics that vary dynamically or through reinforcement learning. Also, it would be better to extend the feature space to more performance measures, including latency, disk I/O, or network utilization, within the scope of performance measures that could be predictive of the models. Collectively, these advances would build the base that this study has laid and bring foretelling auto-scaling nearer to absolutely free, production-scale clouding administrations.

7 Conclusion and Future Work

7.1 Conclusion

In this project a machine learning-based predictive auto-scaling framework of cloud resources management was presented, to overcome the shortcomings of existing reactive scaling methods, including latency, cost inefficiency, and inflexibility of thresholds. The architecture includes three fundamental models LSTM, Prophet, and a Hybrid ensemble to forecast the CPU utilization and actively optimize the instance numbers in an AWS Auto Scaling Group (ASG). It also proposes the mechanisms of cooldown management, log-based observability, GroupInServiceInstances tracking to estimate latency. The outcomes reveal that ML-based scaling is more responsive, respects SLA requirements, and is lower cost to operate than traditional ASG, particularly with dynamic or spiky workloads. The Hybrid model specifically is based on the presence of LSTM and Prophet providing more efficient scaling decisions with a more stable behavior.

7.2 Limitations

Being a promising framework, the proposed one also had limitations in experiment circumstances. The LSTM model was over-sensitive and had longer metric histories (more than or equal to 10 time steps) even though it had a better accuracy. Prophet was strict

and having some surprises or non-seasonal. In addition, the Hybrid model maintained constant weight values (70% LSTM, 30% Prophet) and this might not be the best fixed values in all workloads and when deploying it. Finally, the detection of SLA violations in the process of scale-up delays speaks to the necessity of more intelligent triggering logic or preemptive policies.

7.3 Future Work

Although the framework is reliable in the tested experimental conditions, improvements can be achieved by way of future research. The reinforcement learning approach to adaptive thresholding might enable the system to change its scaling policies over time. Adding real-time anomaly warning would enhance the robustness by blocking unreliable spikes or noisy data. Additionally, it might be possible to integrate with vendor-neutral observability platforms like OpenTelemetry to support a more comprehensive gathering of telemetry and compatibility with hybrid and multi-cloud environments. Finally, making the framework scalable to multi-resource (e.g., memory, I/O) and orchestration in containerized systems such as Kubernetes would increase its applicability to practice-level cloud-native systems. Future research could explore integrating this framework with Kubernetes Horizontal Pod Autoscaler (HPA) or Vertical Pod Autoscaler (VPA) to support containerized applications.

References

Alharthi, S., Alshamsi, A., Alseiari, A. and Alwarafy, A. (2024). Auto-scaling techniques in cloud computing: Issues and research directions, *Sensors* **24**(17): 5551.

Arbat, A., Rajasekharan, M. and Srinivasan, R. (2022). Wasserstein adversarial transformer for cloud workload prediction, *arXiv preprint arXiv:2203.06501* .
URL: *https://arxiv.org/abs/2203.06501*

Barua, M. and Kaiser, G. (2024). Ai-driven resource allocation framework for microservices in hybrid cloud platforms, *arXiv preprint arXiv:2412.02610* .
URL: *https://arxiv.org/abs/2412.02610*

Bensalem, A., Shen, Y. and Elbaz, A. (2023). Scaling serverless functions in edge networks: A reinforcement learning approach, *arXiv preprint arXiv:2305.13130* .
URL: *https://arxiv.org/abs/2305.13130*

Garcia, M. and Patel, A. (2025). Workload prediction for proactive resource allocation in large-scale cloud-edge applications, *Electronics* **14**(16): 3333.
URL: *https://www.mdpi.com/2079-9292/14/16/3333*

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory, *Neural Computation* **9**(8): 1735–1780.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*, OTexts. `https://otexts.com/fpp2/`.

Kyrychenko, A., Smirnov, V. and Tikhomirov, I. (2025). Predictive autoscaling in aws serverless by means of machine learning and sqs metrics, *Proceedings of the CEUR*

Workshop on Cloud Computing Technologies and Applications, Vol. 3963, CEUR Workshop Proceedings, pp. 61–70.
URL: *https://ceur-ws.org/Vol-3963/paper7.pdf*

Lin, W., Zhao, T. and Chen, H. (2025). Online ensemble transformer for accurate cloud workload forecasting in predictive auto-scaling, *arXiv preprint arXiv:2508.12773* .
URL: *https://arxiv.org/abs/2508.12773*

Pintye, I. (2024). Enhancing machine learning-based autoscaling for cloud applications, *The Journal of Supercomputing* .

Qiu, H., He, X., Yu, T. and Sun, Y. (2023). Aware: An extensible reinforcement learning framework for autoscaling in production systems, *USENIX Annual Technical Conference (USENIX ATC '23)*.
URL: *https://www.usenix.org/conference/atc23/presentation/qiu-haoran*

Rubak, A. and Taheri, J. (2023). Machine learning for predictive resource scaling of microservices on kubernetes platforms, *Karlstad University Studies* .
URL: *https://www.diva-portal.org/smash/get/diva2:1869856/FULLTEXT01.pdf*

Suleiman, B. (2023). Predictive auto-scaling: Lstm-based multi-step cloud management, *Proceedings of the International Conference on Intelligent Cloud Computing*, Springer.

Syed, A. A. and Anazagasty, E. (2024). Ai-driven infrastructure automation: Leveraging ai and ml for self-healing and auto-scaling cloud environments, *International Journal of Artificial Intelligence, Data Science, and Machine Learning (IJAIDSML)* **5**(1): 32–43.
URL: *https://ijaidsml.org/index.php/ijaidsml/article/view/81*

Taylor, S. J. and Letham, B. (2017). Prophet: Forecasting at scale, `https://facebook.github.io/prophet/`.

Wu, J. and Singh, P. (2025). Flas: A combination of proactive and reactive auto-scaling architecture for distributed services, *arXiv preprint arXiv:2510.20388* .
URL: *https://arxiv.org/abs/2510.20388*

Zhang, L. and Kim, S. H. (2024). Hybrid reinforcement and time-series learning for predictive cloud auto-scaling, *IEEE Transactions on Cloud Computing* .