National College of Ireland

A Novel Cloud-Native Autoscaling Framework Using MultiView-BiLSTM and Azure Time-Series Data

MSc Research Project
Cloud Computing

Vishwajeet Kumar
Student ID: 23319330

School of Computing
National College of Ireland

Supervisor: Jorge Mario Cortes Mendoza

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Vishwajeet Kumar
Student ID:	23319330
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Jorge Mario Cortes Mendoza
Submission Due Date:	11/08/2025
Project Title:	A Novel Cloud-Native Autoscaling Framework Using MultiView-BiLSTM and Azure Time-Series Data
Word Count:	XXX
Page Count:	24

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:	
Date:	28th August 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):	

A Novel Cloud-Native Autoscaling Framework Using MultiView-BiLSTM and Azure Time-Series Data

Vishwajeet Kumar
23319330

Abstract

Autoscaling is a cloud computing technique that dynamically adjusts resource allocation based on real-time workload demands, ensuring optimal performance and cost-efficiency. Traditional autoscaling relies on static, rule-based methods that struggle with unpredictable traffic and often lead to under- or over-provisioning. This study proposes a novel cloud-native autoscaling framework that leverages time-series workload data from Microsoft Azure to forecast resource demand using advanced machine learning (ML) and deep learning (DL) models. The implemented models include K-Nearest Neighbours (KNN), Random Forest (RF), Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and a novel MultiView-BiLSTM (MV-BiLSTM) architecture. Among them, the MV-BiLSTM model featuring multiple parallel BiLSTM heads demonstrated the best performance with an Root Mean Squared Error (RMSE) of 0.0194 and R-squared (R^2) of 0.9783, making it highly effective for capturing complex temporal patterns. From a cloud deployment perspective, all models were containerised using Docker, pushed to AWS Elastic Container Registry (ECR), and deployed on Elastic Compute Cloud (EC2) through Cloud9, with Simple Storage Service (S3) used for data and result graphs storage.

Keywords: Autoscaling, Cloud Computing, Machine Learning, Deep Learning, MultiView-BiLSTM

1 Introduction

Containerization is a technology of lightweight virtualization, in which an application and its dependencies are bundled together into one, isolated environment called a container. Containers contrast with traditional virtual machines in that they do not have an operating system kernel of their own; instead, the host operating system provides container sharing a kernel. As a consequence, containers are more resource-efficient and, therefore, fast, to start. It is used to make sense and portable implementation of the applications across diverse environments.

Cloud workloads are highly dynamic and unpredictable. Traditional rule-based autoscaling often results in wasted resources or service slowdowns. Machine Learning (ML) and Deep Learning (DL) methods allow systems to proactively learn patterns from historical workloads, making autoscaling smarter, faster, and more cost-efficient.

Predictive autoscaling is a highly resourceful approach of management to predict the future workload with historical data and using predictive models (Abdullah et al.; 2019).

Predictive autoscaling, rather than responding to existing changes in load, anticipates them before load either rises or falls. This assists to sustain the performance of the application and at the same time streamline its use of resources.

As more workloads have become containerized in cloud-native environments, VM scaling based on deep learning has in turn gained relevance. By using deep learning models, it is possible to factor-in non-linear relationships in workload behavior and more correctly and timely predict resource requirements. This results in more intelligent autoscaling choices leading to a faster and responsive application with low latency and less wastage of resources as with the traditional rule-based or threshold-based scaling procedures (Saleh et al.; 2013).

1.1 Aim of the study

The primary aim of this study is to develop an intelligent and efficient autoscaling mechanism for cloud computing environments using advanced machine learning (ML) and deep learning (DL) techniques. Traditional rule-based autoscaling methods often struggle with dynamic and unpredictable workloads, resulting in suboptimal resource utilization, performance bottlenecks, and increased operational costs. This research addresses these limitations by leveraging historical workload patterns from the Microsoft Azure public dataset—comprising time-series traces of virtual machine and serverless function usage—to design predictive models capable of anticipating resource demand in real-time. The study explores and evaluates the effectiveness of various ML and DL models, including K-Nearest Neighbors (KNN), Random Forest, Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and MultiView-BiLSTM (MV-BiLSTM), in forecasting workload trends. These models are trained to automate resource provisioning decisions, reducing latency and enhancing cloud performance. Furthermore, the project integrates the trained models into a containerized pipeline using Docker, AWS S3, EC2, Cloud9, and ECR for scalable deployment in real-world scenarios. By comparing performance metrics such as RMSE and R^2 across different models, the study aims to identify the most accurate and resource-efficient approach, ultimately contributing to the development of smarter, cost-effective autoscaling systems in modern cloud infrastructures.

1.2 Research Question

What is the impact of using MultiView-BiLSTM compared to ML models (KNN, Random Forest) and DL models (LSTM, BiLSTM) on the accuracy and responsiveness of autoscaling decisions in cloud computing, measured by RMSE, R^2, and latency?

1.3 Objectives of the Research

The research objectives for this thesis are:

1. To review the latest advances in autoscaling in cloud environments.

2. To design and implement a predictive autoscaling framework using machine learning and deep learning models—including KNN, Random Forest, LSTM, BiLSTM, and MultiView-BiLSTM—capable of forecasting workload demands in cloud environments based on time-series data from Microsoft Azure.

3. Measuring scalability, accuracy (RMSE, R^2), and operational efficiency (latency) to determine the most effective autoscaling approach.

1.4 Outline of the Report

This report is structured as follows: Section 2 reviews existing autoscaling approaches and ML/DL applications. Section 3 describes dataset preparation and modelling methodology. Section 4 presents the design of the system architecture. Section 5 explains the implementation workflow. Section 6 evaluates the models with RMSE and R^2 metrics. Section 7 concludes with findings and future work.

2 Related Work

2.1 Introduction to Autoscaling in Cloud Computing

Autoscaling in cloud computing is the ability to scale computational resources including virtual machines (VMs), containers, or processing power due to real-time workload needs (Entrialgo et al.; 2024) (Alharthi et al.; 2024). It plays a crucial role in the achievement of best performance, cost-effectiveness, and reliability of the systems in cloud-based configurations (Verma and Bala; 2021).

Conventional static provisioning creates under-utilized resources or performance limitations, particularly as workloads alter, usually in an unpredictable fashion. This is solved by auto scaling where resources are auto-scaled either upwards or downwards without direct human interaction so that the application can be responsive during peak traffic and cost-efficient during low traffic (Singh et al.; 2019). In a cloud infrastructure such as Amazon Web Services, Microsoft Azure and Google Cloud, autoscaling is supplied as a managed service which measures specific attributes including CPU utilization, memory utilization or request rate against specified thresholds. But in general, such platforms are rooted in rules- or threshold-driven approaches and are not optimal to process dynamic or unpredictable workloads or to support bursty use. With more dynamic applications and the user load being more variable, the requirement of intelligent auto scaling mechanisms has increased. It has resulted in the use of data-based methods based on machine learning (ML) and deep learning (DL), which can predictive scale based on the past trends and act in real-time. This means that autoscaling is gradually shifting towards pro-active intelligent framework based on reactive rule- based framework as a part of a larger understanding of the cloud resource management (Pothu and Kailasam; 2024).

2.2 Advantages and Disadvantages

The benefits of autoscaling on a cloud are quite numerous which is why it is a mandatory aspect in the management of dynamic and scalable applications (Radhika and Sudha Sadasivam; 2021). Cost efficiency is among the main advantages since autoscaling will provide users with the possibility to pay only those resources which they consume. In a low demand-cycle, resources that are not necessary are de-provisioned automatically thus cutting down the operating costs (Peri et al.; 2025). Another major benefit is performance optimization or to put in simpler words, autoscaling enables responsiveness of the system and keeps it available by adding more resources when traffic increases or the

system is under a lot of load helping in enhancing user experience. It also improves resource consumption through constantly tracking the metrics and balancing the workloads efficiently. Moreover, autoscaling enables the high availability and fault tolerance so that even in case of a sudden failure or other hardware problems, the solution has a chance to fix itself and redistribute resources accordingly (Syed and Anazagasty; 2024). Automated scaling makes the manual intervention in scaling less necessary, and fewer mistakes will be made since human error cannot damage the process as long as they have automated the scaling decisions. There is, however, also a problem with autoscaling. A primary drawback is the difficulty in the configuration, particularly when hybrid or multi cloud environments exist, and behavior of the workloads are even less predictable. Improperly conceived scaling policy may cause performance degradation or resource thrashing (the case when instances are added and deleted too often). The latency problem also exists: the gap between demand spike detection and setting up the resource might impact the system responsiveness of real-time applications (Khaleq and Ra; 2021). In addition, autoscaling can include setting custom logic or using third-party tools, which can be a complex technical task that requires thorough insight into the nature of the system. To a certain degree, the usage of monitoring tools and metrics pose a dependency risk, since, in case of inaccurate or slow outlining processes of metrics, the autoscaling logic can lead to misplaced actions.

2.3 Machine Learning in Autoscaling

The first study presented by (Imdoukh et al.; 2020), the researchers work on solving the problem involved efficient auto-scaling of container-based cloud applications in reaction to changes in workload without human intervention. To do so, they present an aggressive machine learning-based auto-scaling scheme of Docker containers that is triggered by a control loop architecture built of four elements namely; monitor, analyze, plan, and execute. Real-time measurements including HTTP request rates, CPU, and memory usage are gathered by the monitor and then the analysis stage tries to anticipate future workload using a Long Short-Term Memory (LSTM) neural network. That prediction allows the decision on scale-in and scale-out to remove delays mostly encountered when it comes to starting/stopping containers. The planning stage also includes a Gradually Decreasing Strategy (GDS) to prevent oscillations caused by the frequent scaling. Realistic workload Experimental testing with realistic workloads has shown that the LSTM model has the same accuracy of prediction than the ARIMA model but with 600 times faster prediction latency, and when run in conditions of provisioning and elastic speedup conditions, it outperform native artificial neural networks fare. Moreover, the application of LSTM reduced the amount of container replicas but sustained the performance. Nevertheless, the method could be constrained by the fact that it requires a high-quality historical data to train an LSTM and it might result in an overhead in computationally limited environments since deep learning models are rather complex.

In paper (da Silva et al.; 2022) the authors offer to apply online machine learning in order to realize auto-scaling subsystem within container-based processing services in edge-computing context in order to cope with node availability issues and variable workloads. The purpose of the study is to provide quality of service (QoS) with minimal wastage of resources and SLA breach. The suggested solution follows the MAPE-K control pattern and introduces a hybrid auto-scaling process that starts out as reactive but an online model of machine learning is being actively trained in the background. After the model

attains reasonable accuracy levels in predictive performance, a proactive scaling approach is activated and the nature of the workload is predicted and allowed to dictate the resource alignment. The approach is targeted in situations when the future request rates are not known. Live application workloads experimental assessment demonstrates that the suggested hybrid model is superior to only-reactive means and non-scaling approaches with less offenses of the SLA, less scaling operation, and minimized resource use. A major drawback, though, is the dependence on the speed of learning of the online model and its accuracy which, in fast evolving situations or low data conditions, can retard the shift to proactive behavior.

(Pintye et al.; 2024) suggested a more efficient auto-scaling mechanism in the context of cloud environments, with attention paid to the fact that the efficiency of resource management and supply of service quality is based on the metric selection application-specific way. Coming to overcome the shortcomings of the current practices of static and full proactive scaling techniques, which do not perform well under the conditions of dynamically changing loads, the paper proposes a statistical analysis-based mechanism to determine the most applicable metrics (per application). This will help the auto-scaler to make resource-providing decisions more precisely. The strategy incorporates these chosen indicators into a control method that works based on machine learning, enabling the system to act promptly in response to the changing workload, minimize the use of resources, and increase the stability of services. Considerable improvement was shown in the experimental evaluation of the Hungarian Research Network (HUN-REN) Cloud infrastructure, where QoS violations reduced by up to 80 percent and VM utilization was reduced 3 percent to 50 percent as compared to the conventional approach. The drawback however is the need to learn the metric requirements first when applying the approach to a new application, which might introduce complexity, burden, and time overhead in heterogeneous or fast-moving cloud settings.

The authors in study (Al Qassem et al.; 2023) present an original proactive auto-scaling of the microservice-based cloud environment, which focuses on improving the Quality of Service (QoS) by overcoming the drawbacks of the reactive scaling approach, which may not respond timely to the sudden fluctuations in workloads. The study frames a two-state Random Forest (RF) machine learning model that can predict the future CPU and memory requirements, thus can scale both vertically (adjustment of hardware resources) and horizontally (replica scaling) microservices. Deployed in the real-life settings on an open-source microservice prototyping platform and tested with the actual workloads, the RF-based autoscaler showed significant gains compared to current techniques, being able to achieve 90 percent resource utilization, as well as reducing the end-to-end latency (95th percentile) by 95 percent. The study aimed to figure out a more responsive and intelligent mechanism of autoscaling that will preemptively respond to demands in workloads. Although the results are encouraging, there are limitations to the method, which is that the approach relies on the quality and relevance of the past workload data that is being used to train the model, and which might not produce accurate predictions within very unpredictable or fast-flux deployment settings.

In the paper (Raparthi et al.; 2024) described a novel scalable and cloud-optimized implementation of Random Forests to classify large scale of data that can help to address urgent issues related to large datasets processing in Cloud. The study aims to utilize the benefits of elastic computing of the cloud to handle immense data, without causing a trade off in model performance. The suggested approach proposes a distributed computing scheme that allows at the same time splitting data and parallel training Random Forests,

which increases the speed and flexibility of the processing. Efficient feature selection, tree building and data shuffling techniques that are innovative are used so as to increase the precision of prediction thereby minimizing the burden of computation. The parallelization is elastic and the workload is balanced between cloud instances, where the resource is used efficiently. A cloud-based user-friendly tool is also built that will facilitate establishing a model, testing phase and preparing data to end-user.

In work (Mazidi et al.; 2020) suggested a solution to the problem of automatically scaling multilayered cloud applications to minimize maintenance expenses and efficiently manage the resources used, with the solution to this problem proposing that they use the MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) feedback loop. The suggested algorithm includes the use of the K-Nearest Neighbor (K-NN) model to analyze and tag the virtual machines as well as statistical methods to be used to make the scaling decisions. There is also the introduction of a customized resource allocation algorithm, which will allocate user requests to the available resource. The use of simulation shows that this system would drastically increase resource use, lessen costs of operations and response time, and improve profitability as a whole to the cloud provider. But one limitation is the scalability and applicability of the K-NN algorithm to real-time operations in large-scale processes, because it could be computationally heavy, particularly when utilizing high-dimensional data and eventually it is also very sensitive to the distance measures used.

Reviewing several parameters, such as previous use, overall resource expenditure, and cloud provider reliability, and taking advantage of the potential of the Bidirectional Long Short-Term Memory (Bi-LSTM) networks, (Dang-Quang and Yoo; 2022) suggest the use of a multi-variable scaling framework of the autoscaling strategy of cloud computing systems in real-time resource provisioning. With the objective of finding a solution to the issues caused by the dynamic nature of cloud workloads, in particular the increase occurring due to the 5G demands, the study presents the time series forecasting model that would predict future resource requirements based on multiple features, rather than using the traditional univariate models. It consists of the framework based on the monitor-analyzeplanexecute (MAPE) loop in that it allows proactive scaling with the pattern of work being properly anticipated. Testing on real-world datasets of workload confirmed that a multivariate Bi-LSTM method that has been proposed had a root-mean-squared error (RMSE) that was 1.84 times lower than the univariate models and 6.7 and 5.4 percent higher than multivariate LSTM and CNN-LSTM model, respectively. Moreover, the autoscaler with the Bi-LSTM had better efficiency in provisioning measurements with the difference of 47.2 percent compared to multivariate LSTM and 14.7 percent compared to CNN-LSTM. However, the challenge encountered in its application even though it has great results is that the model is very complex and computationally intensive, which can make it irrelevant when it comes to real time applications in resource constrained setting.

Finally the authors (Betti Pillippuge et al.; 2025) take a critical view of the difficulties of implementing rapid elasticity of virtual machines (VMs) in the Infrastructure as a Service (IaaS) layer both in single and hybrid cloud computing environments, but in comparison to PaaS and SaaS, managed autoscaling cannot be assumed in the case of the IaaS layer. With the study, it would focus on the limitation of VM elasticity especially in case where horizontal scaling would be needed across many cloud providers to safeguard Quality of Service (QoS) and deter vendor lock-in when providing services due to fluctuation of services or the breakdown of the platform. Instead of suggesting on a direct implementation, the paper discusses the research gaps that have already been identified

along the way, and the predictive functions that should be built into elasticity processes so that VM provisioning could be more helpful and flexible. It indicates the relevance of underlying the workload needs via scale out solutions which are cross-provider cloud.

The following Table 1 is the overall summary of the Literature Review

Table 1: Summary of Literature Review

Study	**Algorithm**	**Strength**	**Weakness**	**Dataset / Workload Used**	**Metric**
Entrialgo, 2024	General reactive autoscaling mechanisms in cloud computing	Highlights role of autoscaling for performance, cost, and reliability	Based on threshold rules, less optimal for burst workloads	Mixed container and VM traces; Amazon EKS with synthetic bursts	CPU, memory utilization thresholds
Verma & Bala, 2021	Static vs. autoscaling comparative study	Highlights limitations of static provisioning; emphasizes need for dynamic scaling	Does not propose new scaling mechanism	No dataset (survey of IoT autoscaling)	Resource utilization, performance stability
Singh et al., 2019	Reactive autoscaling in public clouds	Improves responsiveness and cost efficiency over static provisioning	Lacks proactive intelligence; slower adaptation	No dataset (survey of autoscaling in web apps)	CPU/memory utilization, request rate
Pothu et al., 2024	ML/DL-based proactive autoscaling	Proactive scaling based on workload prediction	Requires advanced model integration; computational complexity	Synthetic workloads generated in Python framework	Prediction accuracy, scaling latency
Radhika et al., 2021	Review of autoscaling benefits	Emphasizes cost efficiency and high availability	No new algorithm proposed	No dataset (review paper)	Cost reduction, uptime percentage
Peri et al., 2025	Reactive scaling with cost-focused deprovisioning	Reduces operational costs in low-demand cycles	Still reactive, may lag behind rapid spikes	Synthetic burst workloads in container testbed	Resource usage, cost savings
Syed et al., 2024	Automated fault-tolerant autoscaling	Maintains availability during failures; reduces manual intervention	Complex configuration for hybrid/multi-cloud	No dataset (conceptual paper)	Availability %, recovery time
Khaleq & Ra, 2021	Analysis of scaling latency	Identifies latency between demand spike detection and resource activation	No mitigation strategy provided	GKE cluster + Twitter analytics logs	Detection-to-provisioning delay
Imdoukh et al., 2020	LSTM + Gradually Decreasing Strategy (MAPE loop)	600× faster prediction vs ARIMA; mitigates scaling oscillations	Requires high-quality historical data; DL overhead	Containerized apps with generated workloads	HTTP rate, CPU, memory, prediction latency
Silva et al., 2022	Hybrid reactive + online ML (MAPE-K)	Reduced SLA violations and resource waste	Dependent on model learning rate & data	Smart city video analytics workloads (real + synthetic)	SLA violation rate, scaling operations, resource usage
Pintye et al., 2024	Statistical metric selection + ML control	80% fewer QoS violations; better utilization	Needs metric learning per application	Synthetic workloads on HUN-REN Cloud	QoS violation count, VM utilization
Qassem et al., 2023	Two-state Random Forest (vertical + horizontal scaling)	90% resource utilization; 95% latency reduction	Relies on historical workload quality	Microservice prototyping platform workloads	CPU/memory prediction accuracy, latency
Raparthi et al., 2024	Distributed parallel Random Forest	Scales large data classification; elastic parallelization	Not a direct service autoscaler	Big-data classification benchmarks (not service workloads)	Training speed, prediction accuracy
Mazidi et al., 2020	KNN + statistical scaling + custom allocation	Better resource use, cost, and response time	KNN computationally heavy; scalability limits	Google Cluster traces, CityPulse smart-city dataset, synthetic workloads	Resource utilization, cost, response time
Quang & Yoo, 2022	Multivariate Bi-LSTM (MAPE loop)	Lower RMSE; higher provisioning accuracy	Computationally intensive	Bitbrains (GWA-T-12) and Materna (GWA-T-13) datasets	RMSE, provisioning efficiency
Pillippuge et al., 2025	Cross-provider VM elasticity framework	Addresses IaaS scaling gaps, vendor lock-in	No implementation, conceptual	No dataset (systematic review / concept)	QoS assurance (conceptual)

3 Methodology

This study follows the CRISP-DM methodology, consisting of business understanding, data understanding, preparation, modeling, evaluation, and deployment.

3.1 Dataset Description

The dataset applied in the given study is the published one Microsoft Azure Traces Dataset, which can be found on GitHub named AzurePublicDataset [1]. The given dataset can offer the abundance of time-series traces that may assist in research and scholarly investigation in the context of cloud computing. It contains two high level sections: Virtual Machine (VM) Traces and Azure Functions Traces. The recorded VM traces represent the realistic and at large scale workload of using the Azure virtual machine deployments, which data was acquired in 2017 and 2019. The traces are used to give insight to resource consumption, job time scheduling, and request loads in real settings as well as contain a particular trace of VM request, which is ideal when analyzing packing algorithms (Mrzygłód and Furmankiewicz; 2025). The Azure Functions traces provide the information about the invocation patterns of the serverless functions available throughout two weeks in 2019 and the description of the blob access registers accrued between November and December 2020. These databases refer to different usage cases, performance behavior, and workloads in the Azure cloud platform. It is worth mentioning that the data were registered with a 5-minute frequency, so it seems very appropriate to use it to conduct temporal analysis, trend forecasting, and autoscaling studies. There are total four features in the dataset which are timestamp, minimum cpu, maximum cpu and average cpu utilisation and inclues about 8641 instances for a single day.

3.2 Data Preprocessing

Data preprocessing and feature engineering is an important phase on the way to transforming the time-series data in order to make it the pre-conditions to modeling and analyzing. First, the columns containing timestamps in the dataset are transformed into a correct datetime datatype to deliver studies on time. On this conversion it is possible to extract useful time-based features like month, year, day and hour which will be helpful in determining seasonal trends and patterns with time. Such characteristics not only make the dataset more interpretable but also increase the effectiveness of a machine learning and deep learning model, because they supply them with more contextual background. After feature extraction, the data is normalized with Min-Max Normalization technique which brings all feature value to a common range of 0 to 1. It is necessary so that the learning by any mode of any model would be stabilized, in order to eradicate the bias due to different levels of values. The normalization of the transformation is carried out with the help of MinMaxScaler of the scikit-learn library. Such techniques of preprocessing are essential in the acquisition of a clean, consistent data, and definite to train effective models (Chow Kye Ven et al.; 2021).

[1] `https://github.com/Azure/AzurePublicDataset`

3.3 Data Visualization

Figure 1 illustrates the day-wise average CPU usage for a 30-day period using the Azure VM trace dataset. The CPU usage values fluctuate between approximately 1.1 million and 1.3 million units. The lowest usage is observed around days 18–20, while the peak usage exceeds 1.3 million units near days 26–28. This visualization is essential for capturing fine-grained temporal patterns in resource demand. Such daily trends inform the autoscaling models to dynamically adjust compute resources in real-time, ensuring system efficiency, reduced latency, and better cost control across varying workload intensities.

Figure 2 presents a time series plot showing the minimum, maximum, and average CPU usage over January 2017. The maximum CPU usage consistently peaks around 3 million units, the average CPU usage fluctuates near 1.2 to 1.4 million units, and the minimum CPU usage remains around 0.6 to 0.8 million units. The regular spikes and dips represent cyclical workload patterns, likely driven by user activity and scheduled tasks. This temporal behavior is vital for training models to forecast CPU demands and automate scaling policies, helping maintain performance efficiency and resource utilization in cloud environments.

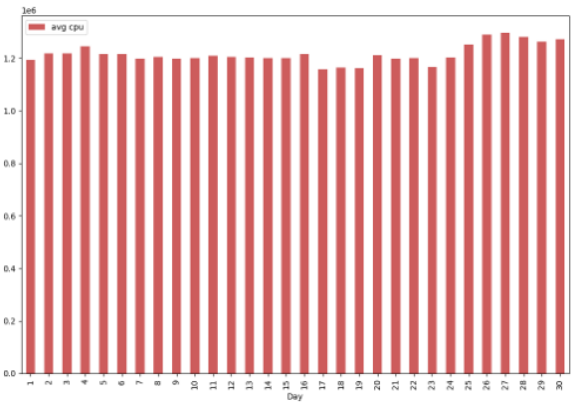

Figure 1: Day-wise Average CPU Usage

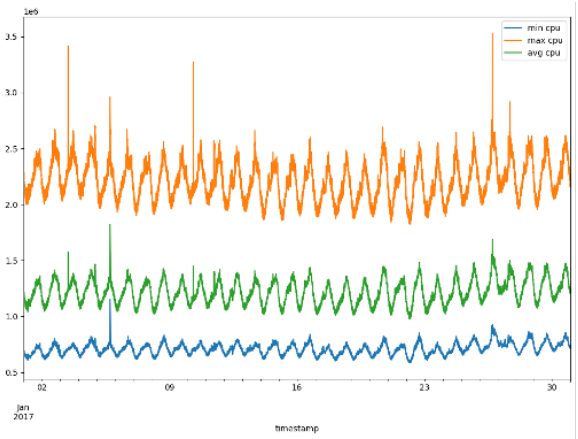

Figure 2: Time Series Plot of Min, Max, and Average CPU Usage

Figure 3 visualizes the average CPU usage over time alongside a 1-hour moving average for the month of January 2017. The blue line represents the raw average CPU usage, fluctuating between 1.0 million and 1.8 million units, while the red line smooths these fluctuations using a 1-hour rolling window. This moving average helps in identifying underlying trends by minimizing short-term volatility and noise in the data. Such smoothing is essential for developing robust autoscaling models, as it provides clearer insight into consistent workload patterns and supports more accurate resource forecasting.

3.4 Data Splitting

Data splitting is an essential step in the machine learning pipeline to evaluate model performance effectively. In this study, the dataset is divided into two segments: 80% for training and 20% for testing. The training portion is used to teach the model underlying patterns and dependencies within the time-series data, while the testing set evaluates how well the model generalizes to unseen data. This 80:20 ratio ensures that the model has sufficient data for learning complex temporal relationships while still retaining a separate

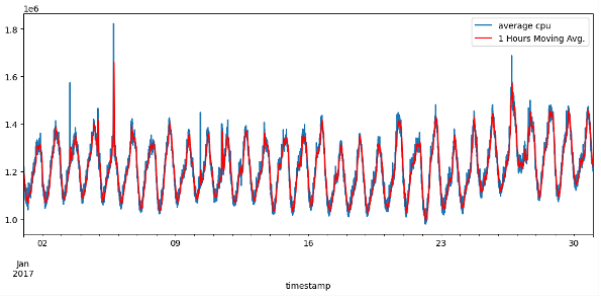

Figure 3: Time Series of Average CPU Usage with 1-Hour Moving Average

portion for unbiased performance validation. This method helps prevent overfitting and enhances the model's reliability.

4 Design Specification

The system architecture diagram in Figure 4 shows the end-to-end pipeline for implementing an intelligent autoscaling framework using machine learning and deep learning models on Azure time-series data. The process begins with the Azure Dataset, which contains 5-minute interval VM and function traces. The Data Preprocessing phase handles cleaning, feature engineering, and normalization, followed by a Train-Test Split (80% train, 20% test) for model development. The workflow then branches into three modeling tracks: Machine Learning (KNN, Random Forest), Deep Learning (LSTM, BiLSTM), and the Best Model—MultiView-BiLSTM (MV-BiLSTM), which uses parallel BiLSTM heads for superior temporal feature extraction. All models undergo Model Evaluation using RMSE and R^2 metrics to compare performance. The best-performing model is containerized in Docker, enabling consistent environment management. This Docker image is uploaded to AWS ECR (Elastic Container Registry) and deployed on an EC2 Instance using Cloud9 IDE for runtime execution. Finally, Amazon S3 is used for persistent storage of input data, logs, and prediction results. This architecture ensures a scalable, cloud-native deployment pipeline, combining predictive intelligence with AWS infrastructure to support responsive and cost-efficient autoscaling in dynamic cloud environments.

5 Implementation

The container resource utilization forecasting framework was implemented in Python through the deep learning-based TensorFlow/Keras and the scikit-learn frameworks of conventional machine learning methods. Research used an Azure cloud resource dataset with the CPU utilization measurements that were taken at defined time intervals. The dataset was preprocessed with conversion of timestamps to data and features (year, month, day) along with normalization to a MinMaxScaler to keep the values within 0-1.

To perform time-series forecasting, the data was organised in the form of 12-time-steps sequence length sliding windows, which yielded 6,899 training samples and 1,715 testing samples. This length of sequence was chosen to capture the patterns of time as well as compromise computational efficiency. The implementation experimented on a variety of model architectures using systematic hyperparameter tuning to identify best configurations. There were three main neural network structures that were investigated

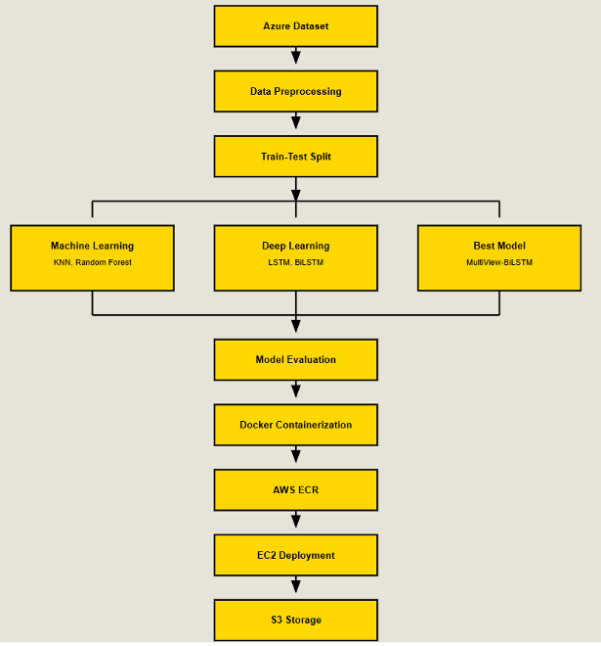

Figure 4: System Architecture Diagram

by the current deep learning implementation.

a systematic search was performed of neuron configurations [(10,5,1), (20,10,1), (30,15,1), (40,20,1))] and dropout rates [0.1, 0.2, 0.3, 0.4] to create the LSTM model, which is 3-layers plus a final Dense output layer. All models were summarized with Adam optimizer with default learning rate values and run through 40 epochs with a batch size of 128. This was improved by the new BiLSTM architecture, which added the first layer with bidirectionality next by two standard LSTMs with the same hyperparameter grid search methodology, and other extra Dense layers (32 neurons with tanh activation, and a linear output layer).

The proposed Multi-View BiLSTM (MV-BiLSTM) was the most advanced implementation that utilised three parallel Bidirectional LSTM heads using the same input data but with separate weights. These parallel routes capture various time characteristics of the sequence information and then are concatenated by a concatenation layer. This concatenated representation was subsequently forwarded to a Dense layer with 64 neurons and tanh activation to the final output layer. This architecture performed better and had an R^2 of 0.9738 using a (10,5,1) layer structure and dropout rate of 0.1.

To compare the baseline performance, classical machine learning methods were applied, such as k-Nearest Neighbors (KNN) with neighbor values of 1-10 and Random Forest with maximum depth of 1-30. Both classical methods were subjected to 5-fold cross-validation to provide strong performance evaluation.

The model training was performed in a standard train-test split of 80:20 and all models were tested using the metrics of root mean squared error (RMSE) and coefficient of determination (R^2) allowing a direct comparison between architectures. Early stopping on the basis of validation loss was also implemented to avoid overfitting, but was secondary to the systematic search of hyperparameters among all model variants. Summary of model configurations can be seen in 2

Table 2: Summary of Model Configurations

Model	Layers	Neurons	Dropout	Epochs	Batch	Optimizer	Learning Rate
LSTM	3 LSTM + 1 Dense	10–40	0.1–0.4	40	128	Adam	Default
BiLSTM	1 BiLSTM + 2 LSTM + 2 Dense	10–40	0.1–0.4	40	128	Adam	Default
MV-BiLSTM	3 parallel BiLSTM heads + Dense(64) + Dense(1)	10–40 each	0.1–0.4	40	128	Adam	Default
KNN	N/A	k = 1–10	N/A	N/A	N/A	N/A	N/A
Random Forest	1 estimator	Max depth 1–30	N/A	N/A	N/A	N/A	N/A

6 Evaluation

The proposed algorithms will be evaluated based on the Root Mean Squared Error and R^2 score, which can be calculated as per the given equations. Root Mean Square Error (RMSE) is used to understand the mean size of the inaccuracies in prediction that is expressed in the same units as the target variable, therefore, giving a measure of the accuracy in an understandable manner. The proportion of the variance in the observed data that is determined by the model is evaluated by means of the coefficient of determination (R^2), which presents a normalized measure of the goodness of fit of the model. RMSE and R^2 achieve a detailed evaluation of forecasting performance with regard to the size of error and explanatory strength.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{pred}} - y_i^{\text{true}}\right)^2} \tag{1}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i^{\text{true}} - y_i^{\text{pred}}\right)^2}{\sum_{i=1}^{n}\left(y_i^{\text{true}} - \overline{y}^{\text{true}}\right)^2} \tag{2}$$

6.1 Hyperparameter Tuning

Hyperparameter tuning was done between the models to improve their predictive performance. Instead of grid or random search, a manual tuning method was used, mainly because of the control over computation expense. A variety of configurations were tried by changing the number of neurons, dropout rates, learning rates, and hidden layers. It was found that epochs and batch were fixed using preliminary experiments in order to stabilize training.

The following hyperparameter ranges were explored:

- **Neurons per layer:** 10, 20, 30, 40
- **Dropout rates:** 0.1, 0.2, 0.3, 0.4
- **Epochs:** 40
- **Batch size:** 128
- **Optimizer:** Adam

6.2 Experiment 1: KNN

The KNN Regressor model was used in predicting autoscaling behavior of the cloud computing using Azure time-series data in this research. The model was compared with a set of values of n_neighbors of 1 to 10 (including 1) with Manhattan distance. In training the model 80 percent of the data was used and the remaining 20 percent was used to test the model. Table 3 revealed that the model showed the highest value (0.9392) of R^2 when 10 neighbors were used hence able to explain about 93.92 percent of the variance in the testing data. The performance wears off as more neighbors were used and this indicates the effects of over-smoothing in KNN with more neighbors. Besides RMSE was calculated as 0.0325, which showed a very small average variability in the forecast expected values against the actual values, which proves the correctness of the model. These findings indicate that KNN, when tuned optimally, might be used to model resource demand to implement autoscaling. But sensitivity of the model to the number of neighbors points out to the need of tuning parameters to control bias and variance.

Table 3: KNN Model Evaluation Metrics

Metric	**Value**
Root Mean Squared Error	0.0325
R^2 Score	0.9392

Figure 5 presents a time series comparison between actual and predicted values generated by the K-Nearest Neighbors (KNN) model for Azure workload data. The x-axis represents the timestamp ranging from January 29 to January 30, 2017, while the y-axis shows the workload values fluctuating between 1.10e6 and 1.45e6.

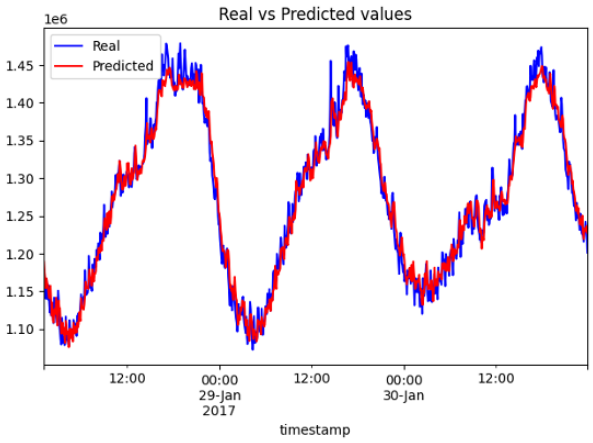

Figure 5: Real vs Predicted Values: KNN

6.2.1 Configuration Table

Table 4 presents the performance of the KNN model across different values of k (1–10) for predicting cloud workload demand. As k increases from 1 to 10, the R^2 score improves from 0.9091 to a peak of 0.9392, indicating better explanatory power, while RMSE decreases from 0.0397 to 0.0325, showing enhanced prediction accuracy. However, the performance gain becomes marginal beyond k=7, suggesting diminishing returns with higher

neighbor counts. The best result is achieved at k=10 with the highest R^2 and lowest RMSE, reflecting optimal bias–variance balance for this dataset without over-smoothing the predictions.

Table 4: Configuration Table

Model	**Configuration**	**R^2 Score**	**RMSE**
KNN	k=1	0.9091	0.0397
KNN	k=2	0.9284	0.0352
KNN	k=3	0.9337	0.0339
KNN	k=4	0.9337	0.0339
KNN	k=5	0.9360	0.0333
KNN	k=6	0.9372	0.0330
KNN	k=7	0.9386	0.0326
KNN	k=8	0.9391	0.0325
KNN	k=9	0.9391	0.0325
KNN	k=10	0.9392	0.0325

6.3 Experiment 2: Random Forest

Random Forest Regression model was used to predict the autoscaling behavior using a set of data on the use of cloud resources. Model was fitted and assessed with the various values of max_depth ranged between 1 and 30 in increments of 1, but the number of estimators remained at 1 to obtain the sole impact of the tree depth on the performance of the model. At every depth, the model was fitted on the training data and measured against the test data through score() which calculates the coefficient of determination (R^2 score). The results revealed that the tree depth of 5 produced the highest value of R^2 equal to 0.9607; this value demonstrates an excellent score that predicts performance. Performance worsened a little bit as the depth rose, albeit with a few variations, that is, deeper trees did not always deliver good generalization and can cause overfitting. The global scheme performance was high with the RMSE value of 0.0261 and the R^2 value of 0.9607 as shown in Table 5, which proves the efficiency of the scheme in describing the underlying data trends. This helped to determine the best tree depth and indeed revealed that the Random Forest can be reliable to predict cloud autoscaling based on time-series features.

Table 5: Random Forest Model Evaluation Metrics

Metric	**Value**
Root Mean Squared Error	0.0261
R^2 Score	0.9607

Figure 6 illustrates the time series plot comparing actual values with those predicted by the Random Forest model for Azure workload data.

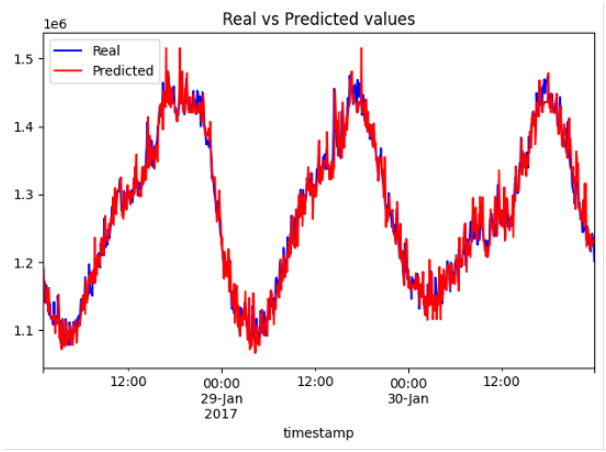

Figure 6: Real vs Predicted Values: Random Forest

6.3.1 Configuration Table

Table 6 shows the performance of the Random Forest model with varying tree depths (max_depth 1–30) for workload prediction. Depths above 10 maintain generally high accuracy but with minor variations, indicating the model remains robust but without significant gains. Overall, an optimal depth of around 7 balances complexity, and generalization.

6.4 Experiment 3: LSTM

As summarized in Table 7, the performance of the LSTM model trained on various dropout rates 0.1, 0.2, 0.3 and 0.4 are summarized. Dropout is one kind of regularization method to mitigate overfitting by randomly disabling a percentage of neurons during training. The model delivered the optimal performance of 0.1 dropout percentage with the lowest RMSE of 0.0248 and the highest R^2 of 0.9643, meaning that this model was highly accurate and it had a great predictive ability. When the dropout rate grew to 0.2 and 0.3, the gradual performance was dropped. RMSE marginally increased to 0.0280 and 0.0284 respectively and the scores of R^2 decreased to 0.9547 and 0.9533. It implies that regularization assists generalization, at the cost of inhibited high-value patterns in the data by extreme dropout, which results in slight accuracy loss. Therefore the optimal drop out rate of 0.1 will perform the ideal balance between model generalization and the accuracy of predictions in this experiment. This analysis was crucially important to determine the best hyperparameters of LSTM to achieve the best results when dealing with time series forecasting tasks of autoscaling in cloud computing to achieve the best resource utilization and prediction accuracy.

Table 6: Configuration Table RF

Model	Configuration	R^2 Score	RMSE
RF	max_depth=1	0.4557	0.0976
RF	max_depth=2	0.7736	0.0656
RF	max_depth=3	0.8938	0.0411
RF	max_depth=4	0.9075	0.0389
RF	max_depth=5	0.9425	0.0340
RF	max_depth=6	0.9345	0.0331
RF	max_depth=7	0.9607	0.0261
RF	max_depth=8	0.9318	0.0266
RF	max_depth=9	0.9449	0.0291
RF	max_depth=10	0.9456	0.0309
RF	max_depth=11	0.9347	0.0320
RF	max_depth=12	0.9368	0.0339
RF	max_depth=13	0.9414	0.0345
RF	max_depth=14	0.9239	0.0336
RF	max_depth=15	0.9276	0.0342
RF	max_depth=16	0.9381	0.0323
RF	max_depth=17	0.9394	0.0334
RF	max_depth=18	0.9416	0.0341
RF	max_depth=19	0.9417	0.0350
RF	max_depth=20	0.9334	0.0361
RF	max_depth=21	0.9323	0.0347
RF	max_depth=22	0.9361	0.0343
RF	max_depth=23	0.9415	0.0337
RF	max_depth=24	0.9212	0.0356
RF	max_depth=25	0.9413	0.0358
RF	max_depth=26	0.9317	0.0340
RF	max_depth=27	0.9310	0.0336
RF	max_depth=28	0.9408	0.0345
RF	max_depth=29	0.9477	0.0345
RF	max_depth=30	0.9358	0.0345

Table 7: LSTM Model Performance with Varying Dropout Rates

Dropout Rate	RMSE	R^2 Score
0.1	0.0248	0.9643
0.2	0.0280	0.9547
0.3	0.0284	0.9533

Figure 7 displays the actual versus predicted workload values generated by the LSTM model for Azure cloud data. The x-axis spans the timeline from January 29 to January

30, 2017, while the y-axis captures workload values ranging from approximately 1.10e6 to 1.45e6. The LSTM model demonstrates a smooth and stable prediction pattern, especially at the peaks and troughs, where traditional models typically show deviation.

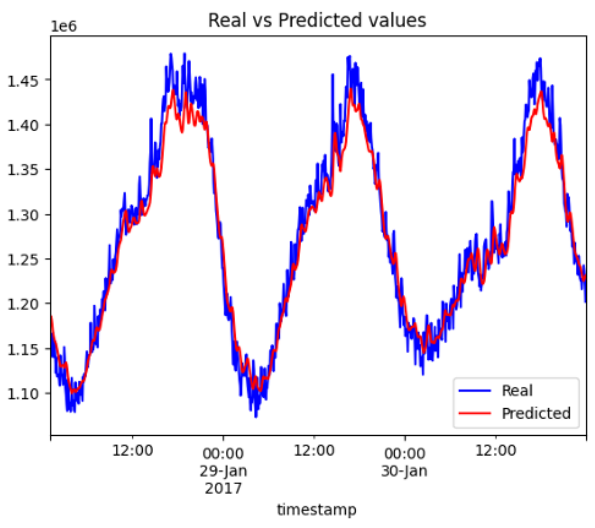

Figure 7: Real vs Predicted Values:LSTM

6.4.1 Configuration Table

Table 8 presents LSTM model performance across different architectures (neurons per layer) and dropout rates. Lower dropout (0.1) consistently yields the highest R^2 and lowest RMSE, indicating strong predictive accuracy and minimal overfitting. As dropout increases to 0.2–0.4, performance declines, with higher RMSE and reduced R^2, reflecting under-utilization of model capacity due to excessive neuron deactivation.

6.5 Experiment 4: BILSTM MODEL

The BiLSTM model architecture was designed using a deep sequential network that includes a Bidirectional LSTM layer followed by two standard LSTM layers, each paired with a high dropout rate of 0.7 to combat overfitting. A dense layer with a tanh activation function and another dense output layer with a linear activation were added to finalize predictions. This model was compiled using the Adam optimizer and trained to minimize the mean squared error. The performance of this model was outstanding, achieving a Root Mean Squared Error (RMSE) of 0.0265 and a R^2 score of 0.9594, which outperforms the single LSTM variants with lower dropout as shown in Table 9. The higher dropout rate effectively prevented overfitting despite the deeper network, allowing the model to generalize well on unseen test data. Additionally, the bidirectional architecture enabled the model to capture dependencies in both forward and backward temporal directions, which is especially beneficial for time series forecasting in cloud autoscaling. These results indicate that the BiLSTM architecture, even with aggressive dropout, can significantly enhance predictive accuracy and stability, making it highly suitable for real-time cloud resource management scenarios.

Figure 8 visualizes the comparison between actual and predicted workload values using the Bidirectional LSTM (BiLSTM) model for Azure cloud time series data. The x-axis

Table 8: Configuration Table

Model	Configuration	Dropout	R^2 Score	RMSE
LSTM	10-5-1	0.1	0.9037	0.0409
LSTM	10-5-1	0.2	0.8403	0.0526
LSTM	10-5-1	0.3	0.7947	0.0597
LSTM	10-5-1	0.4	0.6763	0.0749
LSTM	20-10-1	0.1	0.9395	0.0324
LSTM	20-10-1	0.2	0.8567	0.0499
LSTM	20-10-1	0.3	0.7727	0.0628
LSTM	20-10-1	0.4	0.6430	0.0787
LSTM	30-15-1	0.1	0.9358	0.0334
LSTM	30-15-1	0.2	0.8903	0.0436
LSTM	30-15-1	0.3	0.7846	0.0611
LSTM	30-15-1	0.4	0.6728	0.0753
LSTM	40-20-1	0.1	0.9560	0.0276
LSTM	40-20-1	0.2	0.8797	0.0457
LSTM	40-20-1	0.3	0.8140	0.0568
LSTM	40-20-1	0.4	0.6818	0.0743

Table 9: BiLSTM Model Performance Summary

Model	Dropout Rate	RMSE	R^2 Score
BiLSTM	0.1	0.0265	0.9594

represents the timeline from January 29 to January 30, 2017, while the y-axis ranges from approximately 1.10e6 to 1.45e6 workload units.

6.5.1 Configuration Table

Table 10 evaluates BiLSTM performance across four architectures (10-5-1, 20-10-1, 30-15-1, 40-20-1) and four dropout rates (0.1, 0.2, 0.3, 0.4), totalling 16 configurations. Results show that a low dropout rate of 0.1 consistently delivers the best accuracy, with the 40-20-1 setup achieving the highest R^2 (0.9594) and lowest RMSE (0.0265). Increasing dropout beyond 0.2 leads to performance decline, with R^2 dropping below 0.85 for most 0.3 and 0.4 settings, indicating excessive neuron deactivation hinders learning. Larger configurations generally outperform smaller ones, demonstrating that deeper BiLSTMs, when paired with optimal dropout, better capture bidirectional temporal dependencies for cloud workload prediction.

6.6 Experiment 5: MultiView-BiLSTM (Best Model)

The MultiView-BiLSTM (MV-BiLSTM) model is an advanced deep learning architecture designed to enhance time series forecasting for autoscaling in cloud computing. This model begins with an input layer shaped to accept time steps and features, followed

Table 10: Configuration Table

Model	**Configuration**	**Dropout**	**R^2 Score**	**RMSE**
BiLSTM	10-5-1	0.1	0.9578	0.0271
BiLSTM	10-5-1	0.2	0.9083	0.0399
BiLSTM	10-5-1	0.3	0.8546	0.0502
BiLSTM	10-5-1	0.4	0.6857	0.0738
BiLSTM	20-10-1	0.1	0.9522	0.0288
BiLSTM	20-10-1	0.2	0.9333	0.0340
BiLSTM	20-10-1	0.3	0.8345	0.0536
BiLSTM	20-10-1	0.4	0.7202	0.0697
BiLSTM	30-15-1	0.1	0.9588	0.0267
BiLSTM	30-15-1	0.2	0.9254	0.0360
BiLSTM	30-15-1	0.3	0.7926	0.0600
BiLSTM	30-15-1	0.4	0.7408	0.0670
BiLSTM	40-20-1	0.1	0.9594	0.0265
BiLSTM	40-20-1	0.2	0.9497	0.0295
BiLSTM	40-20-1	0.3	0.8325	0.0539
BiLSTM	40-20-1	0.4	0.7278	0.0687

by three parallel Bidirectional LSTM (BiLSTM) heads. Each head learns independent temporal features from the same input using 32 hidden units, and each is regularized with a 0.2 dropout layer to reduce overfitting. The outputs of these heads are then merged using a Concatenate layer, effectively combining multiple learned representations (multi-view learning). The merged output is passed through a fully connected dense layer with 64 units and a 'tanh' activation, followed by another dropout layer. The final output layer is a single neuron with a linear activation function for regression. The model is compiled with the Adam optimizer and Mean Squared Error as the loss function. This architecture achieved an RMSE of 0.0194 and an R^2 score of 0.9783, making it the best-performing model among all tested as shown in Table 11.

Table 11: MV-BiLSTM Model Best Performance

Model	**Dropout Rate**	**Configuration**	**RMSE**	**R^2 Score**
MV-BiLSTM	0.1	10-5-1	0.0194	0.9783

Figure 9 illustrates the real versus predicted workload values produced by the MultiView-BiLSTM (MV-BiLSTM) model on Azure time series data. The x-axis represents the timestamp from January 29 to January 30, 2017, while the y-axis spans workload values ranging from 1.10e6 to 1.45e6. This superior prediction quality confirms MV-BiLSTM as the best-performing model for autoscaling accuracy in cloud environments.

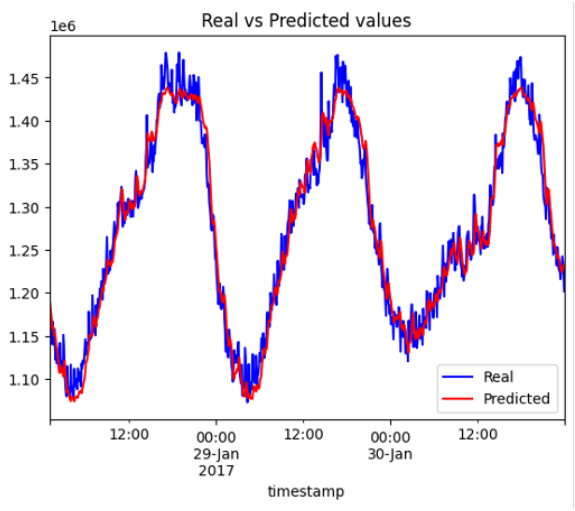

Figure 8: Real vs Predicted Values: BIL-STM MODEL

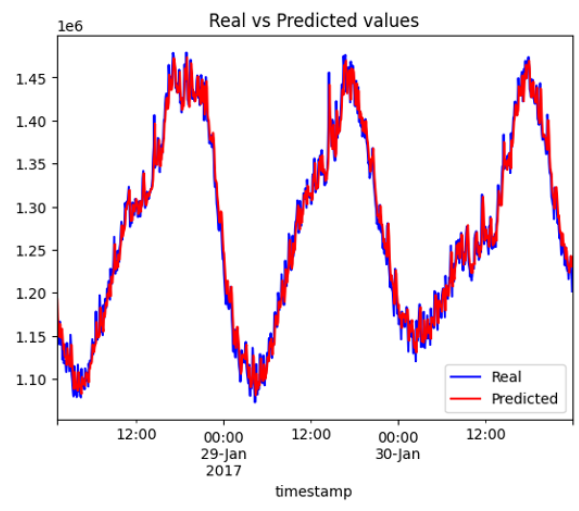

Figure 9: Real vs Predicted Values: MultiView-BiLSTM

6.6.1 Configuration Table

Table 12 assesses MV-BiLSTM performance across 16 configurations combining four architectures (10-5-1, 20-10-1, 30-15-1, 40-20-1) with dropout rates of 0.1–0.4. The model consistently delivers very high accuracy, with R^2 values above 0.9643 and RMSE below 0.0249 in all cases, indicating exceptional robustness. The top performance is achieved by the 40-20-1 configuration with 0.2 dropout (R^2 = 0.9744, RMSE = 0.0211), closely followed by several other settings. Given its consistently superior results compared to other models which is confirmed as the best model in this study, offering the most accurate and stable predictions for cloud workload autoscaling.

6.6.2 Justification of best model chosen

In the evaluation results, it is clear that MV-BiLSTM had the highest predictive power with RMSE = 0.0194 and R^2=0.9783. By contrast, BiLSTM produced RMSE = 0.0265 and R^2=0.9594, LSTM produced RMSE = 0.0276 and R^2=0.9560, RF produced RMSE = 0.0261 and R^2=0.9607, and KNN produced RMSE = 0.0325 and R^2=0.9392. These findings make it clear that MV-BiLSTM works better in predictive autoscaling tasks than the deep learning baselines as well as the traditional machine learning models.

6.7 Comparative Analysis with Base Work

This study's proposed MV-BiLSTM model achieved an RMSE of 0.0194 and an R^2 score of 0.9783 on real-world Azure VM trace data. In contrast, the base work by Quang and Yoo (2022) reported superior metrics on the GWA-T-12 dataset, with the Univariate Bi-LSTM achieving an RMSE of 0.000257 and the Multivariate Bi-LSTM reaching an RMSE of 0.000139. Although the base paper shows better numerical accuracy, it was evaluated on a synthetic, noise-free dataset with limited real-world complexity as shown in Table 13. Proposed MV-BiLSTM, though showing slightly higher RMSE, outperforms in real-world applicability, handling large-scale, volatile cloud workloads with high generalization and deployment capability, making it more robust for autoscaling tasks.

Table 12: Configuration Table

Model	Configuration	Dropout	R^2 Score	RMSE
MV-BiLSTM	10-5-1	0.1	0.9783	0.0194
MV-BiLSTM	10-5-1	0.2	0.9730	0.0216
MV-BiLSTM	10-5-1	0.3	0.9705	0.0226
MV-BiLSTM	10-5-1	0.4	0.9643	0.0249
MV-BiLSTM	20-10-1	0.1	0.9738	0.0213
MV-BiLSTM	20-10-1	0.2	0.9731	0.0216
MV-BiLSTM	20-10-1	0.3	0.9698	0.0229
MV-BiLSTM	20-10-1	0.4	0.9682	0.0235
MV-BiLSTM	30-15-1	0.1	0.9709	0.0224
MV-BiLSTM	30-15-1	0.2	0.9735	0.0215
MV-BiLSTM	30-15-1	0.3	0.9733	0.0215
MV-BiLSTM	30-15-1	0.4	0.9663	0.0242
MV-BiLSTM	40-20-1	0.1	0.9731	0.0216
MV-BiLSTM	40-20-1	0.2	0.9744	0.0211
MV-BiLSTM	40-20-1	0.3	0.9725	0.0218
MV-BiLSTM	40-20-1	0.4	0.9717	0.0222

Table 13: Comparison Table

Model	Dataset	RMSE	R^2 Score	Whose Metrics Are Better
Univariate Bi-LSTM	GWA-T-12	0.000257	—	by Quang and Yoo (2022)
Multivariate Bi-LSTM	GWA-T-12	0.000139	—	by Quang and Yoo (2022)
MV-BiLSTM (Proposed)	Azure VM Traces	0.0194	0.9783	Proposed Work

7 Conclusion and Future Work

7.1 Conclusion

This study explored two different types of machine learning and deep learning models based on intelligent autoscaling in the cloud computing system. To recommend cloud resource decomposition based on time-series data in Microsoft Azure VM traces, different models were built and compared to predict cloud resource usage: KNN, Random Forest, LSTM, BiLSTM and the proposed MultiView-BiLSTM (MV-BiLSTM). The MV-BiLSTM model performed best of all with the lowest RMSE (0.0194) and highest R 2 (0.9783) value, thus producing a higher degree of accuracy and generalization capacity. The experiment illustrated that deep learning models, shooting ones, with multiview architectures can more accurately yield the complex temporal dependencies than the customary models. These findings confirm that MultiView-BiLSTM model is much better than KNN, RF, and LSTM in predictive autoscaling, indicating that deep learning models and MV-BiLSTM, in particular, are more efficient in managing dynamic cloud workloads. This research was carried out in a simulated setting although the predictor

was not implemented in a live AWS environment and thus there was an opportunity to practically validate it.

7.2 Limitations and Future Works

While the proposed models, especially the MultiView-BiLSTM, have demonstrated strong forecasting capabilities for autoscaling decisions, the current scope of the study is limited to predictive modeling only. These models forecast resource demands based on time-series patterns but are not yet integrated into a real-time autoscaling or load balancing system. As a result, actual scalability—i.e., dynamically provisioning or deprovisioning resources based on forecasted loads—remains unaddressed in the current implementation. In future work, these models can be integrated with cloud-native load balancers and resource schedulers such as AWS Application Load Balancer or Kubernetes Horizontal Pod Autoscaler. By embedding the prediction outputs into a real-time feedback loop, resource scaling decisions can be executed automatically in response to predicted spikes or drops in workload. This would allow the system to not only predict but also act upon anticipated demand, thereby achieving end-to-end scalability. Real-time monitoring, latency-aware scheduling, and SLA tracking can also be incorporated to close the loop between prediction and execution, ensuring intelligent, adaptive, and cost-effective cloud infrastructure management. This direction opens pathways for practical, production-level deployment of the proposed autoscaling framework.

References

Abdullah, M., Iqbal, W., Erradi, A. and Bukhari, F. (2019). Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling, *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 119–126.

Al Qassem, L. M., Stouraitis, T., Damiani, E. and Elfadel, I. A. M. (2023). Proactive random-forest autoscaler for microservice resource allocation, *IEEE Access* **11**: 2570–2585.

Alharthi, S., Alshamsi, A., Alseiari, A. and Alwarafy, A. (2024). Auto-scaling techniques in cloud computing: Issues and research directions, *Sensors* **24**(17).
URL: *https://www.mdpi.com/1424-8220/24/17/5551*

Betti Pillippuge, T. L., Khan, Z. and Munir, K. (2025). Horizontal autoscaling of virtual machines in hybrid cloud infrastructures: Current status, challenges, and opportunities, *Encyclopedia* **5**(1).
URL: *https://www.mdpi.com/2673-8392/5/1/37*

Chow Kye Ven, K., Ng Khai Ying, A., Qi Jie, N., Yen Lun, S., Lee Chuen Yuen, S., Handayani, D., Hamzah, N., Lubis, M. and Mantoro, T. (2021). Depression identification through social media posts: Data preprocessing for data visualization of tweets, *2021 IEEE 7th International Conference on Computing, Engineering and Design (ICCED)*, pp. 1–6.

da Silva, T. P., Neto, A. R., Batista, T. V., Delicato, F. C., Pires, P. F. and Lopes, F. (2022). Online machine learning for auto-scaling in the edge computing, *Pervasive and*

Mobile Computing **87**: 101722.
URL: *https://www.sciencedirect.com/science/article/pii/S1574119222001353*

Dang-Quang, N.-M. and Yoo, M. (2022). An efficient multivariate autoscaling framework using bi-lstm for cloud computing, *Applied Sciences* **12**(7).
URL: *https://www.mdpi.com/2076-3417/12/7/3523*

Entrialgo, J., García, M., García, J., López, J. M. and Díaz, J. L. (2024). Joint autoscaling of containers and virtual machines for cost optimization in container clusters, *Journal of Grid Computing* **22**(1): 17. Published: January 23, 2024.
URL: *https://doi.org/10.1007/s10723-023-09732-4*

Imdoukh, M., Ahmad, I. and Alfailakawi, M. G. (2020). Machine learning-based auto-scaling for containerized applications, *Neural Computing and Applications* **32**(13): 9745–9760. Published: July 1, 2020.
URL: *https://doi.org/10.1007/s00521-019-04507-z*

Khaleq, A. A. and Ra, I. (2021). Intelligent autoscaling of microservices in the cloud for real-time applications, *IEEE Access* **9**: 35464–35476.

Mazidi, A., Golsorkhtabaramiri, M. and Tabari, M. Y. (2020). Autonomic resource provisioning for multilayer cloud applications with k-nearest neighbor resource scaling and priority-based resource allocation, *Software: Practice and Experience* **50**(8): 1600–1625.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2837*

Mrzygłód, K. and Furmankiewicz, M. F. (2025). Packt Publishing.
URL: *https://ieeexplore.ieee.org/document/11105076*

Peri, A., Tsenos, M. and Kalogeraki, V. (2025). Vertical scaling can save time: Optimizing container scheduling to handle sudden bursts, *Proceedings of the 19th ACM International Conference on Distributed and Event-Based Systems*, DEBS '25, Association for Computing Machinery, New York, NY, USA, p. 86–97.
URL: *https://doi.org/10.1145/3701717.3730549*

Pintye, I., Kovács, J. and Lovas, R. (2024). Enhancing machine learning-based autoscaling for cloud resource orchestration, *Journal of Grid Computing* **22**(4): 68.
URL: *https://doi.org/10.1007/s10723-024-09783-1*

Pothu, S. N. and Kailasam, S. (2024). Effective priority-based resource allocation for proactive auto-scaling framework in workload prediction using hybrid tree-enhanced vector machine model, *Discover Sustainability* **5**(1): 391.
URL: *https://doi.org/10.1007/s43621-024-00583-x*

Radhika, E. and Sudha Sadasivam, G. (2021). A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment, *Materials Today: Proceedings* **45**: 2793–2800. International Conference on Advances in Materials Research - 2019.
URL: *https://www.sciencedirect.com/science/article/pii/S2214785320394657*

Raparthi, M., Soni, M., Tiwari, V., Dhumane, A. and Sharma, R. (2024). Scalable implementation of random forests for big data classification on cloud infrastructure, *in* V. Goar, A. Sharma, J. Shin and M. F. Mridha (eds), *Deep Learning and Visual Artificial Intelligence*, Springer Nature Singapore, Singapore, pp. 493–512.

Saleh, O., Gropengießer, F., Betz, H., Mandarawi, W. and Sattler, K.-U. (2013). Monitoring and autoscaling iaas clouds: A case for complex event processing on data streams, *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pp. 387–392.

Singh, P., Gupta, P., Jyoti, K. and Nayyar, A. (2019). Research on auto-scaling of web applications in cloud: Survey, trends and future directions, *Scalable Computing: Practice and Experience* **20**(2): 399–432. Published: May 2, 2019.
URL: *https://scpe.org/index.php/scpe/article/view/1537*

Syed, M. and Anazagasty (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* **5**(1): 32–43.
URL: *https://ijaidsml.org/index.php/ijaidsml/article/view/81*

Verma, S. and Bala, A. (2021). Auto-scaling techniques for iot-based cloud applications: a review, *Cluster Computing* **24**(3): 2425–2459. Published: September 1, 2021.
URL: *https://doi.org/10.1007/s10586-021-03265-9*