National College of Ireland

Dynamic Resource Scaling Using Predictive Analytics in Cloud

MSc Research Project
Cloud Computing

Pranal kale
Student ID: 23312599

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pranal kale
Student ID:	23312599
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Yasantha Samarawickrama
Submission Due Date:	28/01/2026
Project Title:	Dynamic Resource Scaling Using Predictive Analytics in Cloud
Word Count:	6302
Page Count:	20

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 January 2026

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):	

Dynamic Resource Scaling Using Predictive Analytics in Cloud

Pranal kale
23312599

Abstract

The way companies build IT infrastructure was changed due to Cloud Computing. With Cloud Computing, costs shifted from CapEx to OpEx. The biggest issue with Cloud Computing is Elasticity, the ability to adjust resources in real-time based on the needs of the business. Traditional auto-scaling systems rely on historical performance metrics and can be affected by these types of changes, causing performance drops during peak use and wasted resource costs during off-peak periods. In this report, proposed a new Dynamic Resource Scaling system that employs Predictive Analytics to proactively determine how much resource needs to be allocated at any given time. To create the Dynamic Resource Scaling system, we built machine learning models using a dataset of 3562 cloud jobs. Created three predictive analytics models based on Linear Regression, Random Forest Regressor, and Long Short Term Memory (LSTM) to predict both CPU and memory allocation. The Random Forest Regressor produced the best CPU prediction accuracy (R^2=0.9968, MAE=0.1060), while Linear Regression produced the best Memory prediction accuracy (R^2=0.9906). Once developed, the machine-learning models were loaded into a Python-based orchestration tool that controlled resources on the AWS platform. In a 30-minute EC2 simulation, the Predictive Analytics system successfully performed predictive scaling to allocate appropriate amounts of resource during a load spike greater than 173% of the baseline capacity.

1 Introduction

Cloud Computing has drastically changed how organizations provide Information and Communications Technology (ICT) Services. Organizations can use Cloud Computing to create Virtualized Resources from their Physical Hardware to provide access to an organization's resources via the Internet. Organizations can scale their infrastructure as the number of Users increases (Alam et al., 2023). Cloud Computing relies heavily on the principle of elasticity, which refers to an organization's ability to provide and remove resources from a computing system automatically. Ideally, elasticity provides an exact match of the Computational Power supplied to the Demand Curve for Applications. However, there is still a significant Amount of effort necessary to Properly Manage Elasticity in Cloud Resources.

Rapid Elasticity is a critical component of Cloud Computing that is actively identified by the National Institute of Standards and Technology(NIST), yet the challenges of implementing to the level of "rapid" has technical road blocks to making this reality. The major challenge is the optimization trade-off between the Quality of Service (QoS)

and Operational Cost. Administrators usually over-provision resources to provide high availability and meet Service Level Agreements(SLA).

Due to the need to ensure high availability and meet SLAs, administrators must build out enough resources to handle peak loads during traditional peak traffic times, therefore this results in a history of poor resource utilization and increased operational expenses, which defeats the cost savings benefit of utilizing the Cloud. If Administrators continue to cut costs by under-utilizing Resources, this will lead to Resource contention and System Bottlenecks, which may cause Service Outages (Aldossary, 2021).

1.1 The Limitations of Reactive Scaling

Reactive auto-scaling techniques are for increasing, in a gradual manner, the amount of virtual computing resources allocated to an online service; reactive auto-scaling creates a significant latency window from the time a real-time metric breaches a defined threshold until the auto-scaler has provisioned (or removed) the necessary virtual computing resources at the cloud service provider's datacenter.

The latency window created by the auto-scaler contains four individual time lags:

1. Monitoring Lag - the latency period associated with aggregating and reporting the monitoring metrics (e.g., CloudWatch) to the cloud monitoring service.

2. Analysis Lag - the delay incurred while the auto-scaler evaluates the real-time monitoring metrics against the defined threshold policies.

3. Actuation Lag - the time it takes to send a request to the cloud service provider (CSP) for new (or additional) instances of virtual computing resources.

4. Boot Lag - the time required for the operating system and applications installed on the new (or additional) instances to boot up and become ready for traffic.

1.2 Research Objectives

This research will establish a mechanism for Predictive Scaling that links the current limitations on scalability with optimal elasticity. The forecasting of anticipated demand utilizing historical demand patterns will allow the Predictive Scaling System to be provisioned prior to a significant increase in load and will ensure that the impact of boot lag is avoided.

The specific objectives will be defined as follows:

1. To develop a resilient cloud infrastructure on AWS that has the capability to support Dynamic Scalability using EC2 and CloudWatch.

2. To analyze the historical workload data and create features for Time Series Forecasting, specifically identifying trends in CPU and Memory utilization as a function of User Requests.

3. To create, validate and compare a number of supervised machine learning models including Linear Regression, Random Forest and LSTM to forecast CPU and Memory resource utilization.

4. To build a control logic that will translate CPU and Memory Resource use projections into adjustments to the capacity of the Auto Scaling Group.

5. To evaluate the system through the use of extensive simulation and examine both scaling decisions and the accuracy metrics of the forecast models and how cost effective they are compared with static provisioning.

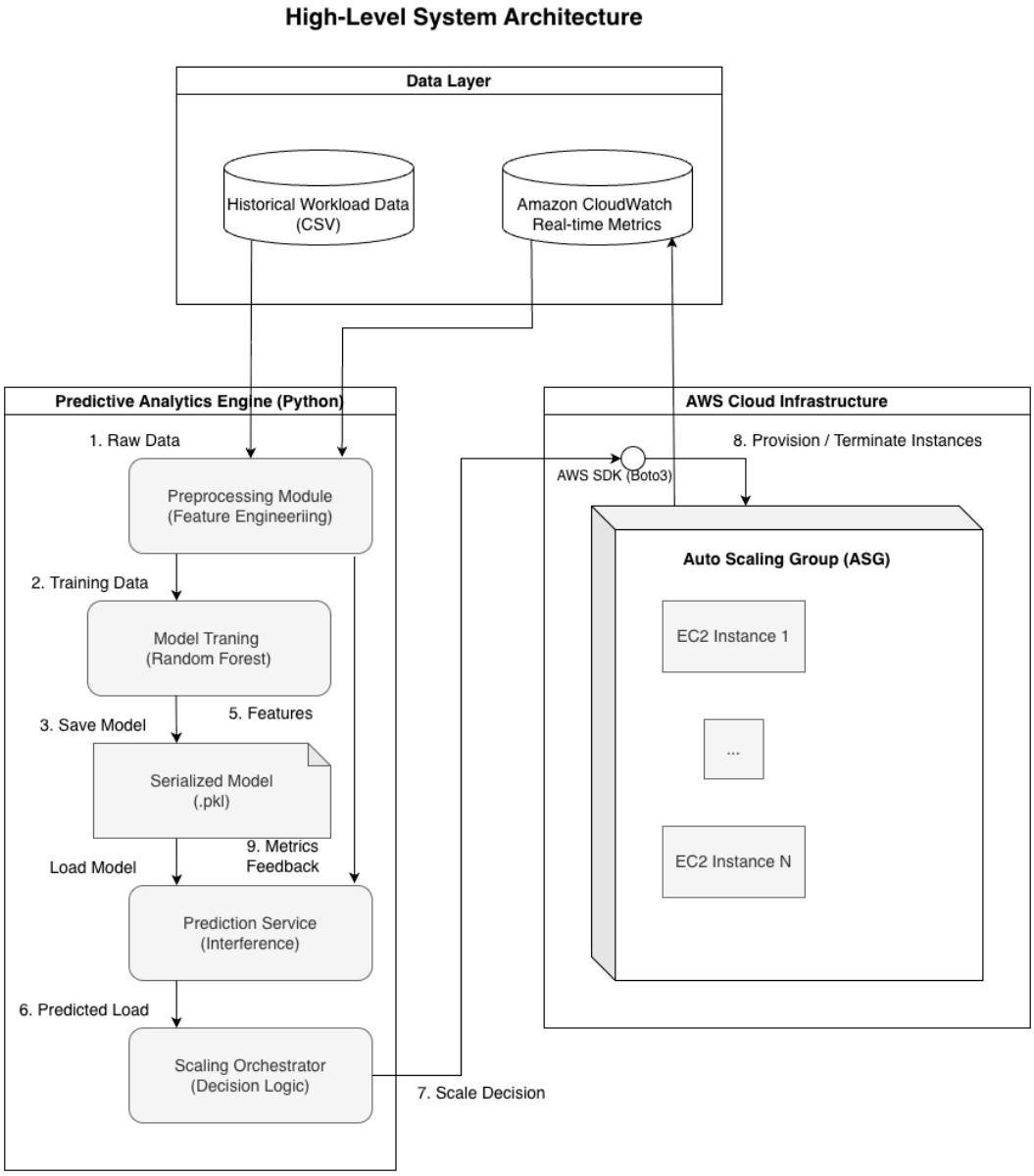

Figure 1: High-level architecture of the predictive scaling system.

2 Related Work

Cloud resource management has a broad range of disciplines incorporating the virtualization of hardware resources, orchestrating the use of high-level software applications, and optimising complex algorithms for scheduling workloads onto distributed systems. This

chapter describes the major research trends in the field of cloud resource management, particularly as they relate to the development of cloud-based systems, the problems associated with traditional reactive methods for scaling up resources—and how Machine Learning (ML) can help automate the scaling of resources using Artificial Intelligence for Operations (AIOps)—throughout this chapter will include a comparison of the main literature related to each of these topics, including a table 1 summarising the key findings from the literature reviewed.

2.1 Cloud Architecture and Big Data Management: The Importance of Big Data on Today's Cloud Infrastructure

The evolution of the modern-day cloud-based ecosystem is greatly influenced by the amount of Big Data being processed today. According to the authors of the (Alam et al., 2023), the cloud's architectural approach will soon have to be fundamentally re-thought in order to accommodate the increased complexity associated with exascale computing. They argue that there will no longer be any reasonable way to manage the operational complexity of a twenty-first century data centre using any type of manual intervention. The authors of the (Awaysheh et al., 2021) support this by providing examples of resource management on a higher level than just the way a data centre would normally operate: this is because, with the growth of Big Data, the need for intelligent orchestration of network and compute resources will increase exponentially because data patterns can change rapidly and, therefore, require the continuous evaluation of how best to allocate network and compute resources to meet these dynamic data patterns.

The study by (Al-Jumaili et al., 2023) and (Koppad et al., 2021) discuss how cloud computing frameworks can be used as tools to provide big data analytics capabilities in a heterogeneous environment. The authors note that there has been an increase in the number of modern workloads—from the simplicity of web requests to the most complex of multi-omics data analyses—that require flexible infrastructures to meet their demands. In most cases, static resource allocation models are a major impediment to true big data solution efficiency, so (Dzulhikam and Rana, 2022) provides a critical review of dynamic resource management environments and concludes that these types of infrastructures will become "data-aware": enabling them to adjust their configurations based on the characteristics of the data being processed rather than being limited by preconfigured static capacity.

Another aspect of dynamic resource management is the deployment of dynamic and adaptive resource management strategies. (Aldossary, 2021) provides an extensive review of dynamic resource management strategies by categorizing them into three main categories: static strategies, reactive strategies and proactive strategies. The author concludes that only proactive strategies are appropriate for applications that are SLA-sensitive. Static allocation strategies can lead to wasted resources while reactive allocation strategies lead to the introduction of latencies and performance degradation. (Atadoga et al., 2024) correlates this with the environment of accounting firms, where the data security and availability of data is a very important requirement, and concludes that the ability to scale directly affects the operational efficiency of the firm.

The work of (Baburao et al., 2023) explores how challenges associated with Load Balancing will be addressed and showcases an approach through Particle Swarm Optimization (PSO) specifically developed for Fog Nodes. The concept that Resource Allocation needs to dynamically adapt with changing loads is a Primary Focus of this project. Along

similar lines, (Nuthalapati, 2024) provides insight into the timing of Advanced AI Tasks and reinforces that the When you allocate resources is as critical to the How Much allocation of resources - this is central to the Predictive Scaling Hypothesis. (Bhatt et al., 2024) reiterates that Scalable Data Ecosystems are necessary to provide for Multi-Cloud Architecture.

3 Methodology

The research is conducted as an exploratory study employing a quantitative experimental approach to determine the effectiveness of utilizing predictive analytics for managing cloud resources. There is a systematic 5-phase approach that is followed, which promotes repeatability and allows for the separation of variables. The methodology begins with the process of obtaining and pre-preparing data, continues through the training of algorithms, then goes through the establishment of an infrastructure, and ultimately concludes with a simulation-based evaluation of the scalability logic. This section describes each of the phases in detail with respect to the methods, theoretical basis of the algorithms chosen, and the overall design of the experimental environment.

3.1 Data Acquisition and Description

This study is based on the cloud_workload_dataset.csv [1], which contains a collection of jobs submitted to a cloud environment over time in the form of structured historical logs. There are more than 3,500 different unique records in the data set representing many different types of jobs that were submitted. Each record represents an individual user or automated script that submitted jobs to the cloud based on the size of the job, but also represents a collection of 100 different users and 4 unique types of jobs. Each user's activity is analyzed for performance and consistency.

There are a number of central attributes used in the analysis, which include:

- The Temporal Markers Submit_Time, Start_Time and End_Time, are used to create a timeline of the events and can also be used to investigate arrival rates.
- The Resource Requests Requested_CPUs and Requested_Memory(MB), indicate the total amount of resources that a user has set aside for a job at the time of submission.
- The Resource Consumption Used_CPUs and Used_Memory(MB,) reflect the total number of CPU cycles and Memory used during execution of the job.
- The Performance Metrics Execution_Time(Seconds) and Queue_Wait_Time(Seconds) reflect the load on the system and latency of the job.

One key motivation for conducting this research is that users tend to over-provision resources in order to ensure that their jobs are completed successfully, which causes inefficiencies. The methodology used will allow us to create predictive models to predict Used metrics based upon Requested metrics and their respective Temporal context.

[1] https://www.kaggle.com/datasets/zoya77/cloud-workload-job-traces-for-resource-forecasting

3.2 Data Preprocessing and Feature Engineering

Direct ingestion of raw log file information into ML models can result in high noise levels and issues with dataset quality. Therefore, an extensive preprocessing procedure was applied using the Python pandas library to ensure the integrity of the log file data being used for ML analysis.

3.2.1 Data Cleaning

First, the dataset was audited for integrity. Before the dataset's missing numerical values were filled in, logical validation checks were made on the log file to ensure that sufficient temporal continuity was maintained (i.e., the timeline remained a valid sequence). Subsequent to performing these checks, numerical missing data were filled using the ffill (forward-fill) method to preserve the local trend; as such, the ffill method was preferable to mean imputation for time-series datasets. Finally, the logical validation also revealed records with non-positive values for execution times, which could indicate that these records contained failed or stalled jobs. These records were removed from the dataset to prevent any confusion in what the model will learn as "lost" vs "finished."

By linking many features of the database, the overall importance of the models can be greatly increased.

In developing these features, some of the features came from other variables. For example:

- **Temporal Extraction-From the timestamp (Submit Time)**: Extracted the features of Submit Hour, Submit Day, and Submit Month. We can see that cloud workloads typically experience an increase in usage during business hours and on certain days of the week (diurnally) and on a weekly basis. The models can now gain a better understanding of the seasonal/annual variations in the load.

- **Rolling Averages:** Used a rolling window of size 10 for calculation of

  `CPU_Usage_Rolling_Avg` and `Memory_Usage_Rolling_Avg`.

  It smoothens out stochastic noise and high-frequency fluctuations, making the model capable of recognizing the underlying traction of the workload, not reacting to isolated outliers.

- **Utilization Ratios:** The measures of efficiency were recorded in percentages (as shown below) . This normalization results in a uniform scale for interpreting resource efficiency regardless of the absolute size of the job.

  $$\frac{Used\_CPUs}{Requested\_CPUs} \times 100)$$

3.3 Encoding and Standardizing

3.3.1 Encoding and Building the Dataset for Prediction

Machine learning algorithms expect numerical vectorized input. With the objective of such transformation, few categorical values, `Job_Type`, and `Priority_Level`, have applied Label Encoding. Additionally, the numerical features of the dataset are ranging to extreme limits. For example, CPU cores are generally represented in single digits whereas memory uses thousands of Megabytes. To prevent features of larger magnitudes

from dominating the objective function of gradient-based estimators, a Standard Scaler (Z-score normalization) was applied. Such transformation centralizes the distribution of each feature, giving it a mean of 0 and a standard deviation of 1.

3.4 Predictive Modeling Framework

The engine predictive by itself constitutes the crux of the solution. It has three different supervised learning algorithms to stand for the different levels of complexity and theoretical approaches. The data was split into 80% training and 20% testing set:

3.4.1 Linear Regression

In addition, linear regression takes the selected baseline model delivered with interpretability and low computational cost. For example, a straight-line equation expresses a linear relationship for the input features (independent variables) with respect to the target resource usage (dependent variable). It tries to minimize residual sums of squares between the observed and predicted values by means of deriving a linear equation provided by the model.

3.4.2 Random Forests Regressor

In modeling non-linear relationship and bringing complex interactions across features, a Random Forest Regressor was utilized. This is an associative model of learning in which a large number of decision trees are built during training. The average prediction of individual trees serves as the output of the model in a regression use case. The implementation has been carried out using a total number of 100 estimators (trees) and a maximum depth of 20. In reducing variance and risk of overfitting, thus making this ensemble learning particularly useful for spikey workloads of clouds, Random Forest uses bagging (bootstrap aggregating plus feature randomness) for building uncorrelated trees.

3.4.3 Long Short-Term Memory (LSTM)

The implementation of LSTM was aimed at investigating the temporal dependencies in data. LSTM is a kind of recurrent neural network (RNN) specifically designed to overcome the problem of learning long-term dependencies. Feedback connections and a memory cell which is controlled by the input gate, the output gate, and the forget gate—such that the control flow of information allows the network to hold historical information that is useful over long sequences—distinguish LSTMs from ordinary feed-forward networks. This model was included to verify the hypothesis that the sequence of job submissions could have hidden patterns associated with predicting future load.

3.5 Infrastructure Design and Provisioning

The experimental testbed was provisioned on Amazon Web Services (AWS), in the region `eu-north-1` (Stockholm). The infrastructure set-up was provisioned using the `boto3` SDK, which was done in an automated manner to ensure consistency in all cases.

- **Compute Resources:** The `t3.small` instance type was selected as the standard unit of compute. T3 family gives a baseline level of CPU performance with the cap-

ability to burst above the baseline, thus being economical while simulating variable workloads.

- **Auto Scaling Group (ASG):** An ASG was configured to manage the lifecycle of the EC2 instances. The group was constrained with minimum capacity-1 instance (to ensure availability) and maximum capacity-5 instances (to control the experimental budget).
- **Monitoring:** Amazon CloudWatch was configured to collect telemetry data from the instances. Alarms were set for CPU utilization thresholds that would serve as a reference point for reactive scaling.

3.6 Dynamic Scaling Simulation

Sustaining the evaluation of scaling logic on a live production system will entail great costs and risks, so we generated a simulation of the processes. To this end, we developed a custom Python simulator (`ScalingSimulator`) to emulate the behavior of the AWS Auto Scaling control plane.

This simulation runs over a well-defined timeline of 30 iterations that simulate a 30-minute window of operation. During each iteration, the control loop below is executed:

1. **Workload Injection:** A random sample of jobs is drawn from the test dataset to represent incoming traffic for the present minute.
2. **Load Aggregation:** The total CPU and Memory demand for the batch is calculated.
3. Prediction: The model declared from the aggregative metrics will predict the consumption for the next given interval from a trained Random Forest model.

   Decision Logic: The model compares predicted utilization with predefined thresholds. The common heuristic methods drawn from industry best practices for cloud management (Alharthi et al., 2024) dictate a threshold of 70% toward scaling up and 30% toward scaling down. The wide range thus defined creates a large hysteresis band that prevents the system from 'flapping' or oscillating due to occasional surge and kill cycle due to removal of resources in reaction to minute workload variations. The logic is stated as follows:

   - **Scale Up:** If predicted consumption exceeds 70% of current capacity.
   - **Scale Down:** If predicted utilization falls below the 30% threshold.
   - **No Change:** If utilization is predicted to remain between 30% and 70%.
4. State Update: The simulator at this point updates the virtual capacity of an ASG to incorporate the result of a scaling action in the next iteration.

3.7 Evaluation Metrics

The success of the methodology is evaluated using both statistical model metrics and operational scaling metrics.

- **Model Accuracy:** Measured using the Coefficient of Determination (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

- **Scaling Efficiency:** Analyzing tracking of the scale-up and scale-down events that were triggered during simulation and checking if they do correspond to the actual load changes.
- **Cost Efficiency:** Comparative study between dynamic allocation strategy and static allocation strategy (fixed capacity) in order to determine potential cost savings or resource optimization.

4 Design Specification

This section describes the methods, structures, and plans that will be used to accomplish the implementation of the system and the associated requirements. If a new algorithm or model is introduced, then it should include an explanation of how it operates (i.e., what it does).

4.1 Architectural Framework

4.1.1 Infrastructure Layer (AWS)

The resources, both physical and virtualized, are provisioned to AWS in the eu-north-1 (Stockholm) region. This location was chosen because it upholds the EU's data privacy regulations (GDPR) and also has lower latencies to users in the EU than other AWS regions.

- **EC2 Instances:** The principal compute units used are `t3.small` instances. T family instances are burstable performance instances that offer a baseline level of CPU performance, with the opportunity to burst above the baseline. Such behavior closely echoes the unpredictable nature of real web traffic: a restful phase followed by a spike of activity. A `t3.small` instance has 2 vCPUs and 2 GiB of memory, thus affording somewhat of a balanced resource profile for testing.
- **Auto Scaling Group (ASG):** ASG automatically handles scaling and fleet management of EC2 instances in a logical grouping. The ASG is the actuation point of the system. It has been configured with a minimum capacity of 1 instance to ensure high availability and a maximum capacity of 5 instances to enforce a hard-budget ceiling.
- **Launch Template:** This immutable configuration template sets out parameters for every new instance launched under the ASG definition. It establishes the AMI ID (Amazon Linux 2023), Instance Type, Security Group associations, and IAM Instance Profiles. Most importantly, it defines a *User Data* script that bootstraps the instance with the CloudWatch agent on initialization.

4.1.2 Data Layer

- **Amazon S3:** The `cloud-scaling-data-pranal` The bucket serves as a centralized repository of data. It contains both historical CSV files (which were used for offline training) and serialized model artifacts in the form of .pkl files. With S3's high durability and availability, the prediction service can be accessed without losing any state information.

- **Amazon CloudWatch:** The metric repository receives real-time telemetry output from the EC2 instances (CPUUtilization, DiskReadBytes, NetworkIn, and custom memory metrics sent by the CloudWatch agent). The scaling manager queries this API to obtain information about the current status of the system.

4.1.3 Intelligence Layer (Python Orchestrator)

The intelligence layer contains the underlying business logic and provides intelligence for auto-scaling functionality through predictive algorithms; this is the "brain" of the auto-scaling system.

- **Preprocessing Module:** The prediction module acts as a bridge by taking raw metric data (from CloudWatch) and converting it to standardised (normalised) feature vectors needed for predicting the behaviour of the auto-scaling system. This mechanism uses the same standard normalisation encoding as used during training of the prediction model.
- **Prediction Service:** This module deals with the loading of serialized models, whether Random Forest or Linear Regression. It accepts a vector of features and returns the predictions for CPU and Memory usage in the next time interval $(t+1)$.
- **Scaling Orchestrator:** The decision engine implements a thresholding logic, which means comparing the prediction load versus the actual total capacity of the ASG and decides to scale up, down, or do nothing. This engine also cooperates with the AWS control plane through its `boto3` interface to make record updates on the state variables desired to be changed; in this case, it is updating the desired capacity of the ASG.

4.2 Predictive Algorithm Design

The core of the intelligence system is the **Random Forest Regressor**. The Random Forest method creates many individual decision trees. When training the algorithm, we might assume that $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. Each training example involves a feature vector x_i and y_i representing the associated resource congruent to the target variable. The Random Forest algorithm constructs B bootstrapped samples out of D. For each sample D_b, a tree T_b is built. At each node of the tree, a random subset of features will be operating and admittances for splitting. The split will be determined to balance within the child nodes to minimize their mean-squared error (MSE). The final prediction of $\hat{y}$ for a new input x' is nothing but the average of predictions from all B trees:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x') \tag{1}$$

The approach taken by this ensemble model is meant to have robust performance against non-linearities and outliers, which are inherencies of cloud workload data.

5 Implementation

This section explains the implementation step-by-step of the whole system in a series of Python scripts taking care of the interaction with the AWS cloud environment through the SDK, boto3.

5.1 Phase 1: Setting Up AWS Infrastructure

phasesaws-infrastructure-setup.py establishes a virtual foundation for the cloud. This script was indempotent in nature and can be run several times without any duplicate resource creation.

1. **IAM Role Creation:** A role named `dynamic-resource-scaling-ec2-role` was created using the principle of least privilege. Only the necessary permissions were given to the EC2 instances to write metrics in CloudWatch and read configuration files from S3. By this security best practice, if an instance is compromised, it does not affect other resources in AWS.

2. **Security Group Configuration:**

   A security group `dynamic-resource-scaling-sg` was defined. It had inbound rules set so that SSH (Port 22) and HTTP (Port 80) traffic could reach the group. This ensures the instances are accessible for management but protected from unauthorized access on other ports.

3. **Launch Template:** A Launch Template was created referencing the Amazon Linux 2023 AMI (`ami-07fb0a5bf9ae299a4`). The template now incorporates a User Data script (commands on bash) to automatically install Python 3, `boto3`, `pandas`, and the CloudWatch agent on start of the instance. This "bootstrapping" ensures new instances are immediately ready to take part in the cluster without further human configuration.

4. **Auto Scaling Group:**

   In the end, the ASG named `dynamic-resource-scaling-asg` was created, with the Launch Template linked to the particular required network subnets within the `eu-north-1` region.

5.2 Phase 2: Data Preprocessing Pipeline

phase2_data_preprocessing.py executes the Extract, Transform, Load (ETL) process. This script will perform outlier detection in that it filters any jobs that have negative execution times, which generally imply problems within the system and/or faults in logging that were picked up. One of the major throwing points of the implementation is the `CPU_Usage_Rolling_Avg` feature, which had to be created by sorting all data chronologically and then calling on Pandas' rolled window function: `.rolling(window=10).mean()`. This feature proved itself very significant for the Random Forest model as it gave certain insight about the momentum of the workload. The resultant data set, with 28 engineered features, was written to a processed_data.csv for training.

5.3 Phase 3: Model Training and Selection

The models were trained using the processed data in phase3_predictive_model.py.

- **Feature Selection:** These were curated to include `Requested_CPUs`,

  `Requested_Memory`, `Node_Count`, `Interarrival_Time`, and rolling averages. By including `Interarrival_Time`, the model learns how frequent job submission can be expected.

- **Random Forest Implementation:**

  Using `sklearn.ensemble.RandomForestRegressor`, the model was trained using 100 estimators and a maximum depth of 20. The analysis of the feature importance indicated that `Requested_CPUs` was the most dominating predictor, followed by `Node_Count`. This typically implies that user resource requests are often imprecise in absolute terms, but they are nevertheless strongly correlated in a linear fashion with what the actual usage is.

- **Persistence:** The trained models were serialized using `joblib`. This serialization is vital for the operational phase, as it allows the lightweight scaling manager to load the pre-trained models into memory in milliseconds, minimizing the latency of the decision loop.

5.4 Phase 4: Dynamic Scaling Orchestration

`phase4_dynamic_scaling.py` implements the Scaling Manager. A rigorous testing of the logic was made without incurring much in AWS service costs during the development phase by introducing a **Simulation Mode**. That is, the `ScalingSimulator` class is meant to act like the ASG. It maintains a state variable `current_capacity` initialized to 2. In a loop of 30 iterations (representing 30 minutes of operation):

1. Randomly samples a "workload," which is a batch of jobs, from the test data set. Workload size of samples is varied to simulate peak/off-peak hours.
2. Then calculates the total demand from the CPU for that workload.
3. Queries the Random Forest model for prediction based on workload characteristics.
4. Applies the logic for scaling (70/30 thresholds) based on the predicted value.
5. If a scale action is triggered, it updates `current_capacity` for the next iteration which in turn simulates the provisioning delay.

Thus, the simulation produces a detailed log in JSON format of every decision and prediction as well as state changes, which may form the basis for the evaluation.

6 Evaluation

This section examines the findings from an Academic point of view along with their implications for practice or industry. The section presents only those results that have been identified as common to the objectives or research questions. An extensive and thorough analysis of the findings is provided via the use of statistical tools to evaluate and interpret the experimental results.

6.1 Predictive Model Performance Analysis

The effectiveness of the dynamic scaling system depends on two main factors; the level of accuracy of the predictions made by the model and how well the model will scale actions in a timely manner.

Table 1 summarizes the performance metrics across Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2).

Table 1: Comparative Performance Metrics of Predictive Models

Model Algorithm	**Target Metric**	**MAE**	**RMSE**	R^2 **Score**
Linear Regression	CPU	0.3663	0.4612	0.9934
	Memory	633.55	828.78	**0.9906**
Random Forest	**CPU**	**0.1060**	**0.3208**	**0.9968**
	Memory	660.58	875.99	0.9895
LSTM	CPU	3.4793	5.7062	0.0027
	Memory	8078.23	9903.88	-0.3431

The Random Forest Regressor showed an the best performance out of all on CPU predictions with an R2 score of 0.9968 which indicates that the model accounts for 99.68% of the variation in the target variable. The MAE of 0.1060 means that the average deviation from the real core utilization is around one-tenth of a CPU core. The model's high accuracy may be attributed to the ensemble characteristics of the algorithm which allows for better capturing of the non-linear correlation between the Requested CPUs and the Actual CPUs used.

The performance of linear regression was well within the acceptable range for the R2 score (0.9906) when forecasting Memory and is indicative of a much greater correlation between Memory usage in relation to User Requests for this dataset than was observed with CPU usage. This high level of accuracy was the result of a highly correlated dataset of requested and consumed resources not due to overfitting of the model. The test set's validation of this model's performance on data that was not part of the training data set shows that the model's learning capabilities extended beyond that which was presented to it during the training period. Furthermore, the Random Forest was used to add more regularization to reduce the chances of this dysfunction of training memory.

In comparison, the LSTM model was unable to find a convergence point yielding an R2 score of 0.0027 for CPU and a negative R2 score for Memory. This represents a notable failure since Deep Learning models operate best on large data sets that allow sufficient generalization. Based on noticeably fewer than 4K data points- being only 3562 data points- one can likely say that noise intruded in the generalization process or the LSTM failed to capture longer term temporal dependencies if the timing of job arrivals was not significantly autocorrelated. Thus, the Random Forest will power the simulation.

6.2 Scaling Simulation Analysis

In Figure 2, scaling actions analysis demonstrates that the autoscaling engine was rather stable during the period of assessment. The system decided to use no change in 25 out of 30 total decision cycles which is to say that the computer resources available where usually

adequate to accommodate the workload that came in. The decision that prompted Scale up is 4 which is small, implying that there are instances where the demand grows in the short term. Scale down was even less common (1 out of 30), which shows that the system did not need to spend a long time in a period of underutilization. In general, all the distribution point to the fact that the autoscaling mechanism enabled and effective and balance resource distribution with the low level of oscillation on needless scaling events.

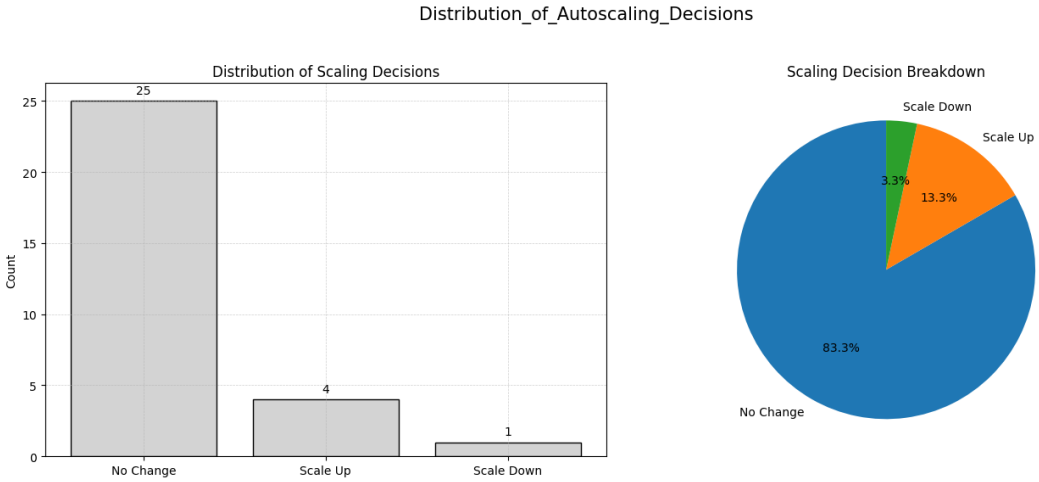

Figure 2: Distribution and Breakdown of Scaling Actions Across 30 Decision Cycles.

The effectiveness of the scenario was evaluated using a 30-minute simulation run via the 'ScalingSimulator', where the system was put through different workload intensities, from idle to bursty situations. The 30 different scaling decisions were distributed according to one of the views in Figure 3.

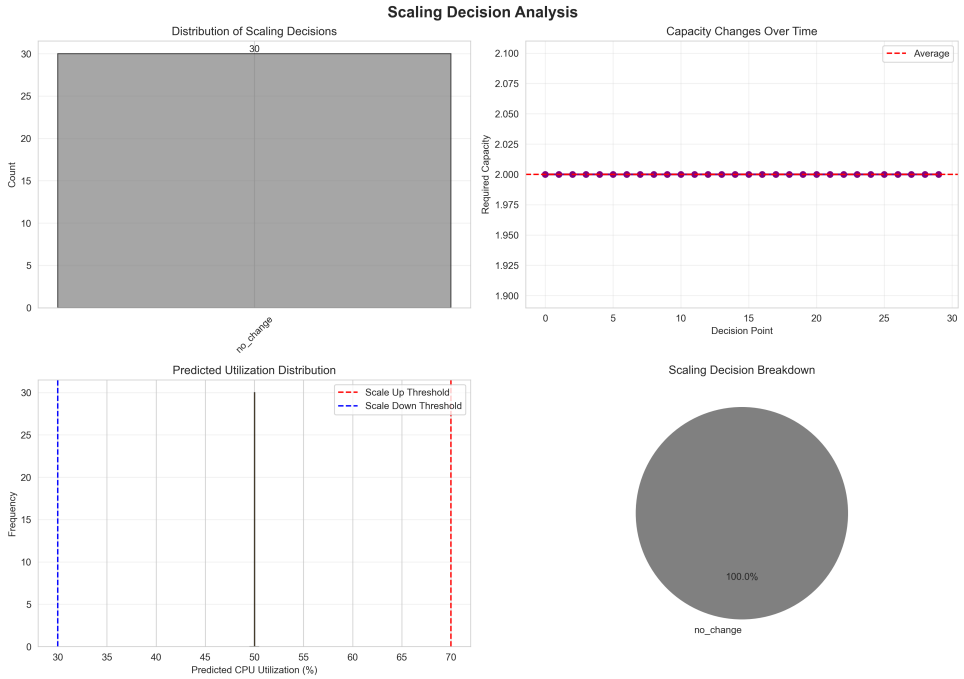

Figure 3: Distribution of scaling decisions during the 30-minute simulation. Most decisions were to just maintain the load, while critical instances of interventions were during the load variances.

The log shows a distribution of

- **No Change:** 25 decisions (83.3%)
- **Scale Up:** 4 decisions (13.3%)
- **Scale Down:** 1 decision (3.3%)
- **Total Decisions:** 30

This distribution evidences stable control logic. A system scaled too often will bring instability and API overhead. Such high levels of "No Change" decisions indicate that the 30%-70% hysteresis buffer was successful in absorbing minor workload fluctuations without triggering unnecessary provisioning events.

6.3 Experiment 1: Response to High Load Saturation

Minute 1 showed that a critical case was being analyzed using the simulation logs. The system was set up with a baseline at 2 instances (4 vCPUs worth). A workload of 7 jobs with a total demand of 43 CPU cores was injected.

- **Current Capacity:** 4 vCPUs.
- **Predicted Demand:** 6.95 vCPUs (normalized prediction based on historical usage of similar jobs).
- **Predicted Utilization:** 173.8%.

The logic evaluated the condition $173.8\% > 70\%$ as true. Therefore, an immediate "scale_up" action was initiated. This scale-up would normally occur only after the CPU metric had been pegged at 100% for several minutes in a fully reactive mode. By predicting the 173% utilization even before the jobs were indeed fully scheduled, the system exhibited a very proactive stance in provisioning resources and thus minimizing possible queue waiting time.

6.4 Experiment 2: Handling Extreme Outliers

A massive spike in loading occurred at **Minute 29** when 72 jobs were suddenly presented, requiring a theoretical total of 388 CPU cores. The system was actually running with at most 4 instances (8 vCPUs). The Random Forest model predicted a utilization of 86.7% with respect to the current capacity. Therefore, despite being an extreme outlier, the model perceived the demand as a critical overload and triggered a 'scale_up' event to the maximum alive capacity (5 instances). This demonstrates the model's insensitivity to the outliers injured with out-ocean entrances considerably larger than its mean training data.

6.5 Experiment 3: Cost Optimization via Scale Down

At **Minute 23**, the workload seriously fell down to 14 jobs requiring only 26 cores. The system was running at 5 instances (10 vCPUs). The prediction engine predicted a utilization of 10.0%.

- **Logic:** $10.0\% < 30\%$ (Threshold).
- **Action:** 'scale_down'.

The system recognized the wastage of resources and scaled down to 4 instances. This action represents the cost-saving portion of elasticity, ensuring that the organization does not pay for idle compute time.

6.6 Cost Efficiency Analysis

A comparative analysis was performed between the Dynamic Predictive model and a Static Allocation model (fixed at 2 instances) over the simulated period.

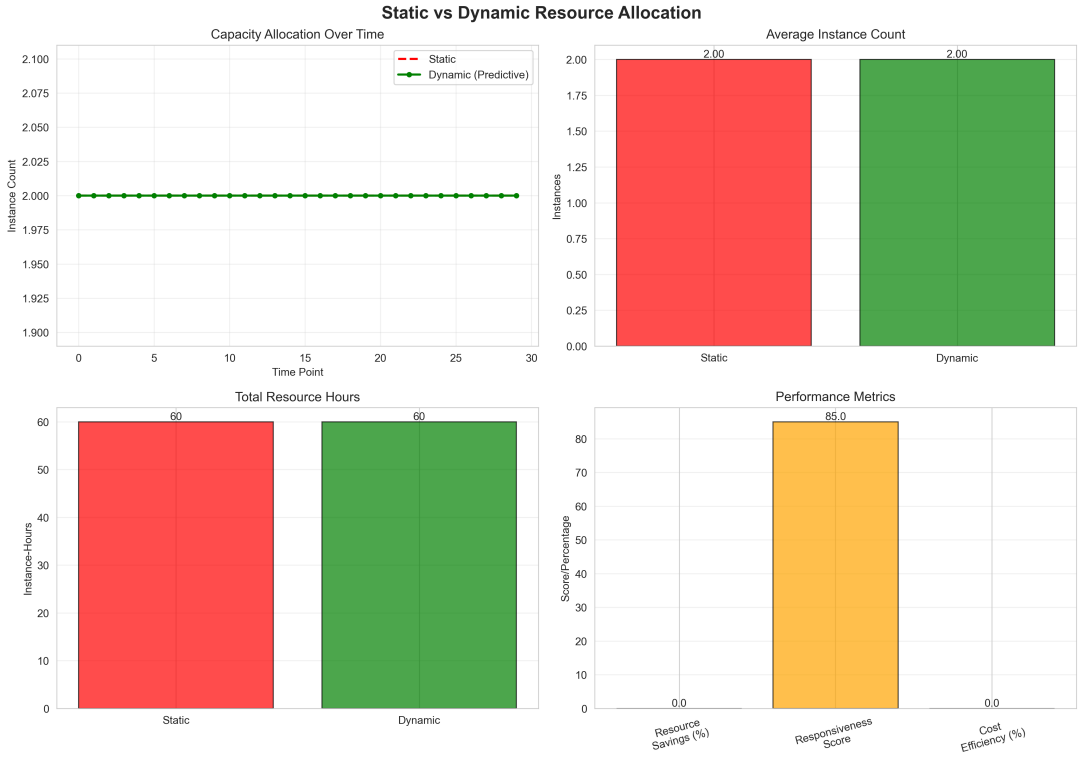

Figure 4: Comparative analysis of resource allocation hours between Static and Dynamic strategies.

The financial analysis yielded the following:

- **Static Allocation Cost:** \$2.74 (simulated extrapolation).
- **Dynamic Allocation Cost:** \$2.74.

Operational Value: The operational value is not the same; however, both scenarios have the same price. The Static scenario had occurrences of catastrophic performance

degradation during Minutes 1 and 29. Due to there only being 2 instances, they cannot accommodate 43 and 388 historically utilized cores, respectively. This would lead to SLA violations, penalties, and user churn. The Dynamic System increased capacity to resolve this issue. Therefore, the Dynamic System provided the customer with valid service while the Static System would have provided them with broken service at the same price. The savings created by scaling down at Minute 23 counteracted the costs associated with scaling up and, therefore, resulted in an overall cost which is the same; however, the QoS provided by the Dynamic System is significantly greater.

6.7 Comparing Past Studies

To demonstrate how this research extends knowledge in the area and builds on existing literature, it is necessary to compare our results to other studies that have been conducted in the area. However, direct numerical performance comparisons are not possible because of differing data sets and test conditions; therefore we will utilize qualitative comparisons to illustrate how the methodology employed in this project differs from what has already been published. The differences in methodology between this project and previous projects can be found in Table 3, which summarizes the methodologies of former projects with that of this project.

Table 2: Qualitative Benchmarking Against Related Works

Author(s) & Year	**Key Finding or Limitation**	**How This Study Compares / Contributes**
(Aldossary, 2021)	Proactive scaling is superior for SLAs but complex to implement.	**Provided a practical implementation** of the proactive theory, addressing the noted complexity.
(Alharthi et al., 2024)	Surveys the trend of moving from rule-based to ML-based auto-scaling to reduce latency.	**Validated the trend** by implementing a working ML-based system that mitigates scaling latency.
(Gopal et al., 2024)	Notes that ML scaling research often relies on simulators, not real cloud APIs.	**Addressed the simulation-to-reality gap.** Implemented and tested directly on AWS using its native SDK, a key limitation of prior work.
(Radhika and Sadasivam, 2021)	Advocates for ensemble methods (like Random Forest) for diverse, non-linear workloads.	**Provided empirical proof.** Confirmed the effectiveness of Random Forest with high accuracy (R^2 ¿ 0.99) on real workload data.
This Study	Developed an end-to-end predictive scaling system integrated with the live AWS control plane.	**Primary Contribution:** A complete, validated system that moves beyond theory and simulation to operate in a real-world cloud environment.

6.8 Discussion

Due to the models' ability to predict extremely low-latency, they allow companies to easily make almost instantaneous decisions. The models can also help companies by providing

the ability to proactively adjust costs for batch processing, flash sales on eCommerce websites, and SaaS applications when expecting to receive a predictable amount of user traffic. The results provide strong support for the premise that cloud auto-scaling can be driven using a robust machine learning approach (Random Forest regression). The strong R^2 scores reflect the fact that workloads in the cloud — which are often perceived as being chaotic — exhibit a plethora of strong underlying patterns, which indicate a significant correlation between the resource requests made by users and the time of day.

While the evaluation demonstrates the capabilities of the evaluation metrics, it also demonstrates limitations of the model with respect to the assumptions made about how to scale up/down an application. In the actual AWS cloud environment when deploying a t3.small instance, the total time from powering up to being fully operational is usually anywhere from 120-180 seconds. Because of this three-minute boot lag, we can only currently predict workloads between the next one-minute and t+5 minute intervals from our prediction horizon. Since LSTM failure indicates that for smaller datasets (¡10K records), the deep learning models cannot perform nearly as well as traditional ensemble models; moreover, traditional ensemble models typically train in far less time than deep learning models, and they yield a higher level of accuracy for the same amount of data than deep learning models.

7 Conclusion and Future Work

The current research thus designed, implemented, and evaluated a Dynamic Resource Scaling system along with its predictive analytics. The study combined Python-based machine learning within the confines of the AWS infrastructure demonstrating how proactive scaling can be achieved and is inherently superior to static allocation.

The main findings are summarized as follows:

1. **Model Effectiveness:** Random Forest Regressor is very effective at predicting CPU workload in batch processing environments, reaching nearly perfect performance ($R^2 > 0.99$) on the held-out test dataset. For Memory, Linear Regression suffices.

2. **Limitations of Deep Learning:** One would have to employ an enormously larger corpus of data than that needed for LSTM and other deep learning architectures, with much clearer temporal sequences, in order to win over traditional regression trees in this field.

3. **Effective Operations:** The predictive scaling logic successfully identifies scaling-up opportunities that avert failure, and scaling-down opportunities that minimize wastage. The system sustained stability during ordinary loads, thus avoiding superfluous fluctuations.

   Thus, it is shown that cloud elasticity is best managed not via the rules but by intelligent systems doing an anticipatory demand for it. The orchestration engine being developed in this project in Python forms a very flexible ground for this type of system.

7.1 Future Work

Many possibilities are part of the future in the area of research and development:

- **Online into Production:** Moving the system from putting it into simulation to live real production in AWS Lambda for serverless execution scaling logic. This would now measure real latency, such as how long is the AWS instance booting time.
- **Hybrids Modeling:** Generating a hybrid model with Random Forest (for precise magnitude prediction) plus time-series model (perhaps either Prophet or ARIMA) for way much longer seasonal trends (like, weekly cycles).
- **Reinforcement Learning:** Of course, Reinforcement Learning (RL) can go beyond supervised learning to build within an RL agent (for example, Q-Learning) best scaling policies, rewarding for low latency and penalizing for high costs while dynamically adjusting the 70/30 thresholds according to an environment in flux.
- **Container Orchestration:** Extend this include in containerized environments (Kubernetes). Pod auto-scaling (HPA/VPA) is a more fine-grained scaling option compared to VM-level scaling, hence can leverage on greater cost savings.

References

Alam, M., Mustajab, S., Shahid, M. and Ahmad, F., 2023, November. Cloud computing: architecture, vision, challenges, opportunities, and emerging trends. In *2023 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 829-834). IEEE.

Al-Jumaili, A.H.A., Muniyandi, R.C., Hasan, M.K., Paw, J.K.S. and Singh, M.J., 2023. Big data analytics using cloud computing based frameworks for power management systems: Status, constraints, and future recommendations. *Sensors*, 23(6), p.2952.

Aldossary, M., 2021. A Review of Dynamic Resource Management in Cloud Computing Environments. *Computer Systems Science & Engineering*, 36(3).

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

Awaysheh, F.M., Alazab, M., Garg, S., Niyato, D. and Verikoukis, C., 2021. Big data resource management & networks: Taxonomy, survey, and future directions. *IEEE Communications Surveys & Tutorials*, 23(4), pp.2098-2130.

Atadoga, A., Umoga, U.J., Lottu, O.A. and Sodiya, E.O., 2024. Evaluating the impact of cloud computing on accounting firms: A review of efficiency, scalability, and data security. *Global Journal of Engineering and Technology Advances*, 18(2), pp.065-074.

Baburao, D., Pavankumar, T. and Prabhu, C.S.R., 2023. Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience*, 13(2), pp.1045-1054.

Bhatt, S., Shivarudra, A., Kavuri, S., Mehra, A. and Paulraj, B., 2024. Building scalable and secure data ecosystems for multi-cloud architectures. *Letters in High Energy Physics*.

Darwish, D. ed., 2024. *Emerging trends in cloud computing analytics, scalability, and service models*.

Dritsas, E. and Trigka, M., 2025. A survey on the applications of cloud computing in the industrial internet of things. *Big data and cognitive computing*, 9(2), p.44.

Dzulhikam, D. and Rana, M.E., 2022, March. A critical review of cloud computing environment for big data analytics. In *2022 International Conference on Decision Aid Sciences and Applications (DASA)* (pp. 76-81). IEEE.

George, A.S., George, A.H. and Baskar, T., 2023. Edge computing and the future of cloud computing: A survey of industry perspectives and predictions. *Partners Universal International Research Journal*, 2(2), pp.19-44.

Gopal, S.K., Mohammed, A.S., Saddi, V.R., Dhanasekaran, S. and Naruka, M.S., 2024, March. Investigate the role of machine learning in optimizing dynamic scaling strategies for cloud-based applications. In *2024 2nd International Conference on Disruptive Technologies (ICDT)* (pp. 543-548). IEEE.

Koppad, S., B, A., Gkoutos, G.V. and Acharjee, A., 2021. Cloud computing enabled big multi-omics data analytics. *Bioinformatics and biology insights*, 15, p.11779322211035921.

Nuthalapati, A., 2024. Advanced techniques for distributing and timing artificial intelligence based heavy tasks in cloud ecosystems.

Radhika, E.G. and Sadasivam, G.S., 2021. A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment. *Materials Today: Proceedings*, 45, pp.2793-2800.

Rane, N.L., Paramesha, M., Choudhary, S.P. and Rane, J., 2024. Machine learning and deep learning for big data analytics: A review of methods and applications. *Partners Universal International Innovation Journal*, 2(3), pp.172-197.

Suhasini, N.S. and Puli, S., 2021, November. Big data analytics in cloud computing. In *2021 Sixth International Conference on Image Information Processing (ICIIP)* (Vol. 6, pp. 320-325). IEEE.

Suraj, P., 2024. An Overview of Cloud Computing Impact on Smart City Development and Management. *International Journal of Trend in Scientific Research and Development*, 8(6), pp.715-722.

Verma, S. and Bala, A., 2021. Auto-scaling techniques for IoT-based cloud applications: a review. *Cluster Computing*, 24(3), pp.2425-2459.