Machine Learning-Driven Strategies for Efficient Resource Management in Cloud Data Centers

Mustafa Daraghmeh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

February 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis

prepared By: **Mustafa Daraghmeh**

Entitled: **Machine Learning-Driven Strategies for Efficient Resource Management in Cloud Data Centers**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Yuhong Yan

_____ External Examiner
Dr. Abdallah Khreishah

_____ Arm's Length Examiner
Dr. Rodolfo Lima Coutinho

_____ Examiner
Dr. Jamal Bentahar

_____ Examiner
Dr. Glitho Roch

_____ Supervisor
Dr. Anjali Agarwal

Approved by _____
Dr. Jun Cai, Graduate Program Director

March 21, 2024 _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Machine Learning-Driven Strategies for Efficient Resource Management in Cloud Data Centers

Mustafa Daraghmeh, Ph.D.

Concordia University, 2024

Cloud computing is one of the major paradigms in the information technology industry, offering diverse scalable on-demand services over the Internet. Nevertheless, managing and predicting workloads in cloud data centers is a challenging task due to the dynamic nature of cloud services. In order to reduce costs and improve performance while managing cloud resources efficiently, it is essential to obtain highly accurate projections and estimations. Therefore, this thesis proposes a methodological framework that integrates multiple machine learning models to improve estimation accuracy and enable better decision-making within cloud data centers. In terms of clustering, we develop segmentation pipelines that incorporate various clustering techniques with different data preprocessing methods to improve the cloud workload segmentation process. This process aims to reveal hidden patterns within workloads to obtain segmentation based on various data-driven perspectives. In predictive modeling, we delve into the enhancement of prediction precision, focusing on single-output and multi-output forecasting models. For single-output-based prediction, we propose a multilevel learning-based model for resource utilization prediction that leverages anomaly, clustering, and ensemble methods to improve prediction outcomes. Also, we present a proactive regression-based cost estimation approach, navigating the complexities of prediction-based cloud service pricing and the effect of various target transformation methods on prediction accuracy. In addition, we propose a host load prediction, leveraging both imbalance and ensemble learning methods to improve prediction and handle the challenge of the imbalance states within cloud computing systems. For multi-output-based prediction, advanced predictive models are proposed to forecast function invocation patterns at the user, application, and function levels within serverless computing

environments. In this thesis, we conducted research that utilizes advanced data analysis techniques, including windowing, dimensionality reduction, and ensemble learning, to enhance the robustness and precision of workload segmentation and predictive models within cloud environments. We evaluated the proposed models based on their efficiency in processing real cloud workloads using various performance metrics. The findings of this thesis hold the potential to revolutionize cloud resource management, leading to more intelligent, adaptable, and cost-effective cloud operations.

Acknowledgments

I would like to extend my heartfelt gratitude to all individuals who have provided support and encouragement throughout my PhD journey. I am especially indebted to my thesis supervisor, Dr. Anjali Agarwal, whose unparalleled mentorship and profound support have been the keystone of my academic and personal growth. Dr. Agarwal has been a guide and a catalyst for my intellectual development, challenging me to push beyond apparent limits and fostering a spirit of scholarly independence. Her belief in my potential has been a driving force behind my accomplishments.

I would like to extend my appreciation to my advisory committee, Dr. Giltho Roch, Dr. Jamal Bentahar, Dr. Rodolfo Lima Coutinho, and Dr. Abdallah Khreishah, for their insightful feedback and guidance. I wish to express my sincere gratefulness to Dr. Yaser Jararweh, who mentored me during my master's and continued to offer his invaluable advice and support during my PhD journey. I would also like to thank NSERC CRD and MITACS, in partnership with Cistech, for their substantial financial assistance, which has been a crucial cornerstone of my research. I am profoundly thankful for their investment in my work.

I express my profound gratitude to my late mother (Aysheh) and late sister (Ibtisam), whose wisdom and lessons persistently steer and motivate me. This academic achievement serves as a lasting testament to their legacy. The unwavering devotion, patience, and belief in my goals demonstrated by my immediate relatives (father, brothers, and sisters) have consistently acted as my main driving force. I am infinitely appreciative of their selfless sacrifices and unwavering support. Finally, a heartfelt acknowledgment goes out to my wife, Aseel, and my children, Bana, Rashed, and Naya. Your love, resilience, and cheerfulness have been my sanctuary. The sacrifices you have made through this challenging and rewarding journey are deeply recognized and appreciated.

Contents

List of Figures

List of Tables

List of Acronyms

SLA Service Level Agreement

QoS Quality of Service

CRMS Cloud Resource Management System

SDM Selection and Decision-Making

ML Machine Learning

VCI Virtual Computing Instance

SVM Support-Vector Machine

BPNN Back-Propagation Neural Network

SC Silhouette Coefficient

CHI Calinski-Harabasz Index

DBI Davies-Bouldin Index

VMM Virtual Machine Manager

CM Container Manager

PCA Principal Component Analysis

SP Standard Pipeline

PP Poly Pipeline

FaaS Function-as-a-Service

ARIMA AutoRegressive Integrated Moving Average

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

DBSCAN Density-Based Spatial Clustering of Applications with Noise

MAE Mean Absolute Error

MSE Mean Squared Error

RMSE Root Mean Squared Error

R^2 Coefficient of Determination

MAPE Mean Absolute Percentage Error

KDE Kernel Density Estimation

CS Combined Score

VM Virtual Machine

SVM Support-Vector Machine

CSPA Cluster-based Similarity Partitioning Algorithm

HGPA HyperGraph Partitioning Algorithm

MCLA Meta-CLustering Algorithm

WAT Wasserstein Adversarial Transformer

SDS Software-Defined Systems

KNN k-Nearest Neighbors

TT Training Time

CBR CatBoost Regressor

GBR Gradient Boosting Regressor

ETR Extra Trees Regressor

LGBM Light Gradient Boosting Machine

RFR Random Forest Regressor

XGB Extreme Gradient Boosting

DTR Decision Tree Regressor

ENet Elastic Net

Lasso Lasso Regression

LR Linear Regression

LAR Lasso Least Angle Regression

Ridge Ridge Regression

BRidge Bayesian Ridge

AdaBoost AdaBoost Regressor

DR Dummy Regressor

OMP Orthogonal Matching Pursuit

Bi-LSTM Bidirectional Long-Short Term Memory

TP True Positives

TN True Negatives

FP False Positives

FN False Negatives

kappa Cohen's kappa

MCC Matthews Correlation Coefficient

TPR True Positive Rate

FPR False Positive Rate

ROC Receiver Operating Characteristic

AUC Area under the ROC Curve

SMOTE Synthetic Minority Over-sampling Technique

RT Resampling Time

IoT Internet of Things

VANET Vehicular Ad-hoc Network

AI Artificial Intelligence

MSVR Multi-output Support Vector Regression

ICSA Immune Clone Selection Algorithm

TPPs Temporal Point Processes

HPC High-Performance Computing

Chapter 1

Introduction

1.1 Cloud Computing and the Quest for Efficiency: Overview

Cloud computing represents a significant paradigm shift in the way businesses and individuals access and use computing resources. This paradigm provides scalable, on-demand access to vast cloud services, ranging from software applications to storage and computing power, delivered over the Internet [1]. Pay-per-use models, governed by Service Level Agreement (SLA), allow cloud users to avoid substantial upfront development costs and operational expenses associated with maintaining physical infrastructure. This flexibility and cost-effectiveness make cloud computing an integral component of various emerging technologies such as the Internet of Things (IoT), Vehicular Ad-hoc Network (VANET), e-health, and extensive data management [2], [3].

Central to cloud computing's efficiency is virtualization technology. It bridges operational environments and the underlying hardware, offering a versatile approach to provisioning and managing various Virtual Machine (VM)s and containers on heterogeneous physical machines [2]. This adaptability is crucial to optimize resource usage and minimize operational costs.

An essential strategy to improve resource and energy efficiency in cloud data centers is dynamic workload consolidation. This approach consolidates workloads within fewer hosts by reallocating them from underutilized hosts to other fitted ones, reducing energy consumption and related costs [4]. However, balancing effective consolidation while maintaining high Quality of Service (QoS) is challenging, as overconsolidation can affect the performance of computing systems [5].

Machine Learning (ML) techniques such as regression, classification, clustering, and anomaly detection are crucial in improving the efficiency of cloud computing. By incorporating highly accurate models, we can optimize various aspects of cloud operations, such as efficiently allocating resources, detecting and addressing system overloads and underloads, and making informed decisions [6]. The dynamic learning and adaptation capabilities of the ML models to the ever-changing cloud environment significantly improve the performance of cloud data centers.

However, cloud computing continues to evolve, driven by the need for more efficient, cost-effective, and sustainable operations. Integrating highly precise ML models represents the forefront of this evolution. These advances aim to meet the growing demands of users and applications while ensuring adherence to the QoS standards. As this field continues to grow, the quest for efficiency in cloud computing remains a central theme that shapes the development of new strategies, technologies, and innovative ML models.

1.2 Problem Statement

The resource capacities of cloud data centers are enormous but not boundless. A typical data center comprises thousands of servers with varying characteristics. Inefficient resource management increases cloud operational costs. Therefore, elastic resource management approaches are needed to optimize resources and energy efficiency. On-demand resource provisioning remains the most significant challenge in developing robust and adaptable techniques based on intelligent models. Dealing with resource overprovisioning and underprovisioning problems is necessary to determine the most resilient elastic resource management solutions [7].

The problem of resource overprovisioning arises when the allocated resources required to carry out users' application needs exceed the actual demands, as shown in Figure 1.1 (a). This condition may ensure that applications run without SLA violations. However, it remains a significant issue that increases resource waste and incurs additional costs. On the other hand, a resource underprovisioning problem occurs when the available resources are insufficient to fulfill the application's needs, resulting in unexpected SLA violations and performance degradation within the computing systems. Figure 1.1 (b) shows this state, where the shaded area indicates a lack of resources relative

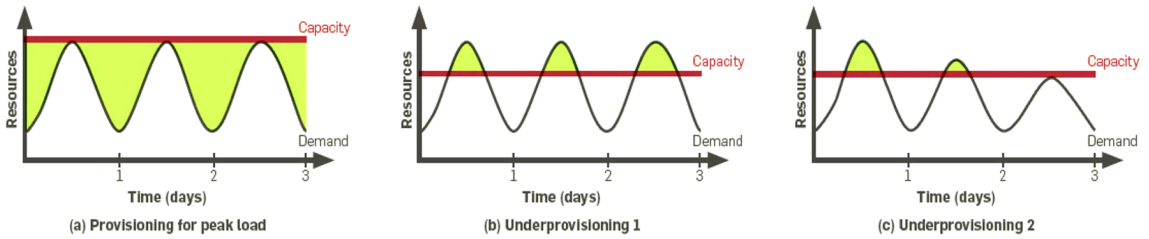

Figure 1.1: Resource over and under provisioning problems [7].

to the needs of the running applications. This problem also risks losing income from users who may permanently opt out of services due to poor QoS experiences over time, as seen in Figure 1.1 (c).

Cloud data centers comprise highly dynamic workloads with various usage patterns. This variety can result in imbalanced server loads, resulting in hot and cold spots within the data center [2]. The Cloud Resource Management System (CRMS) should be equipped with the appropriate procedures to identify and collect information on active workloads to avoid these potential problems. Various ML models can be used to improve selection and decision-making within data centers. However, a data preprocessing pipeline should be applied to the data collected from various sources before creating ML models for more reliable results. It is a crucial step because it affects the learning process of the ML models, in which the inadequate pipeline results in incorrect interpretation [8]. Cloud workload is characterized by a high fluctuation load containing multiple hidden patterns that change over time, complicating the knowledge discovery process. Thus, a proper data preparation pipeline will improve data quality and provide valuable insight for efficient model learning, which turns into adequate estimations that can be incorporated into the decision and selection modules of CRMS to achieve efficient cloud resource management.

In conclusion, cloud data centers face significant challenges in effectively managing and predicting workloads, a crucial aspect for optimizing performance and cost efficiency. One central area of concern is the segmentation of cloud workloads, where traditional clustering techniques often fall short due to the dynamic and multifaceted nature of cloud environments. Furthermore, predicting various operational parameters, such as resource utilization, cost, and load balancing, is increasingly complex. This complexity stems from the rapidly evolving and heterogeneous nature of cloud infrastructure, which requires more sophisticated predictive modeling techniques. Both single-output

and multi-output predictive models are under scrutiny to enhance their accuracy and efficiency. In serverless computing, specifically, the need for efficient predictive methods to manage function invocation patterns is paramount, given the intricate interplay of user, application, and function levels in these architectures. The overarching problem is thus developing a framework that incorporates advanced clustering and predictive modeling techniques that can effectively navigate and harness the complexities inherent in cloud computing environments.

1.3 Research Motivation

In an era where cloud computing serves as the foundation for a wide range of applications, the dynamic nature of cloud environments poses unique challenges. This thesis investigates advanced clustering and predictive modeling techniques to address these challenges, emphasizing the importance of highly accurate and adaptable models in cloud operations.

The segmentation of cloud workloads using sophisticated clustering methods is critical for gaining detailed insights into resource management. Given novel models that aim to adapt to changing workloads while revealing hidden patterns from multiple perspectives is essential. Such advancements are required for effective resource monitoring and strategic management in cloud environments. Equally important is the evolution of predictive modeling. Traditional approaches often need to be more generalized in the complex world of cloud operations. This thesis advocates for novel multilevel and multi-output predictive models incorporating various analytical techniques to accurately forecast critical aspects like resource requirements, cost estimation, and load prediction. These models go beyond traditional forecasting by being more generalized and enabling more robust proactive strategies to operate cloud services smoothly.

Using ML models is critical in implementing the aforementioned advanced techniques, enabling intelligent strategies driven by data. The rich open-source libraries, such as scikit-learn [9], combined with powerful scientific libraries such as [10] and SciPy [11], support a wide range of algorithms for regression, classification, anomaly detection, and clustering. These open-source libraries with comprehensive documentation enable the creation of intelligent models for CRMS, which can transform workload management through predictive analytics and insightful data interpretation.

In summary, the motivation behind exploring novel advanced clustering and various predictive modeling techniques in cloud computing is driven by the imperative to incorporate management operations with highly accurate models that align with the dynamic nature of cloud workloads. This thesis emphasizes the potential for ML and advanced analytical techniques to revolutionize cloud resource management. By leveraging these technologies with high precision, we can improve cloud management operation efficiency and pave the way for more intelligent, adaptable, generalized, and efficient computational models to meet the ever-changing demands.

1.4 Research Objectives

The main goal of this research is to enhance the decision-making efficiency of cloud operational management through integrating distinct high-precision clustering and predictive modeling techniques into the CRMS. The primary objectives are as follows.

(1) **Develop Advanced Clustering Techniques for Cloud Workload Segmentation:** The primary objective is to revolutionize traditional clustering methodologies by exploring and revealing hidden patterns from multiple perspectives during the data preprocessing stage instead of using a fixed preparation pipeline. By doing so, we aim to create novel adaptable models for the dynamic and complex nature of cloud workloads, enhancing segmentation efficiency and applicability considering different forms of data. Moreover, incorporating a novel ensemble clustering method that considers different clustering schemes from different perspectives leads to adequate partitioning of cloud workloads with more reliable clustering results.

(2) **Enhance Predictive Modeling Accuracy in Cloud Computing:** The imperative objective is to improve the precision of single-output predictive models in different aspects within cloud data centers by incorporating various analytical analysis and ML techniques. This objective includes developing novel and sophisticated models that incorporate multilevel learning and analysis methods for predicting resource utilization requirements, cost estimation, and host load status, addressing the heterogeneous and evolving nature of cloud infrastructure.

(3) **Optimizing the Prediction of Invocation Patterns in Serverless Computing:** The main

objective is to enhance the simplicity and efficiency of invocation pattern predictions within serverless environments through multi-output regression models. These models are employed to predict function invocation patterns at different levels, such as user, application, and function. We aim to deliver accurate predictions by incorporating various analytical analyses, enabling more generalized models and better decision-making at each level of analysis.

(4) **Integrate Advanced Data Analysis Techniques:** The primary objective is to analyze the influence of techniques such as windowing, dimensionality reduction, multilevel and ensemble learning, and various data preparation methods on improving predictive model performance. This objective delves into the various techniques to manage complex and large-scale data within cloud environments. We aim to provide a comprehensive understanding of the roles of these techniques and how they can be leveraged to optimize the performance of the prediction and estimation models with decision-making systems in the cloud data centers.

(5) **Contribute to the Body of Knowledge in Cloud Computing:** The main objective is to provide valuable insights and methodologies to both academic and industry fields. We are focused on presenting new benchmark models that can help improve the operational management of cloud workloads and predictive modeling within the cloud data centers, thereby enabling the further development of cloud decision-making systems in various settings.

The objectives mentioned above collectively desire to address crucial challenges in cloud computing by pushing the limits of current capabilities in workload management and predictive modeling. The research aims to establish more effective, adaptable, generalized, and cost-effective models for the operational management of cloud computing solutions.

1.5 Methodological Framework

The proposed methodology operates in multiple interconnected layers to ensure a cohesive and elastic CRMS. The framework combines continuous monitoring with intelligent decision-making enabled by advanced learning models incorporating workload segmentation and various predictive modeling methods driven by data. Below is an elaborate explanation of the framework elements.

(1) **Cloud Subscribers Interface and Monitoring Component:** Cloud subscribers engage with cloud services using a variety of interfaces. The CRMS relies on the usage patterns and requests as a primary source to form operational actions. The REST API is a line for communication between cloud subscribers and cloud resources. The monitoring component tracks all real-time subscriber interactions, resource usage, and system health. The centralized database stores all monitored data, serving as a repository for analytics and decision-making processes.

(2) **Learning Models:** This central component enables intelligent decision-making within CRMS. It consists of diverse ML models that execute tasks such as clustering, anomaly detection, classification, and regression. These models analyze the collected data from various to identify usage patterns, predict future demands, and detect irregularities.

(3) **Global Manager:** This component acts as the central brain of the methodological framework of CRMS, interpreting the insights derived from the learning models. It makes informed decisions about resource allocation and manages workload distribution across the cloud data

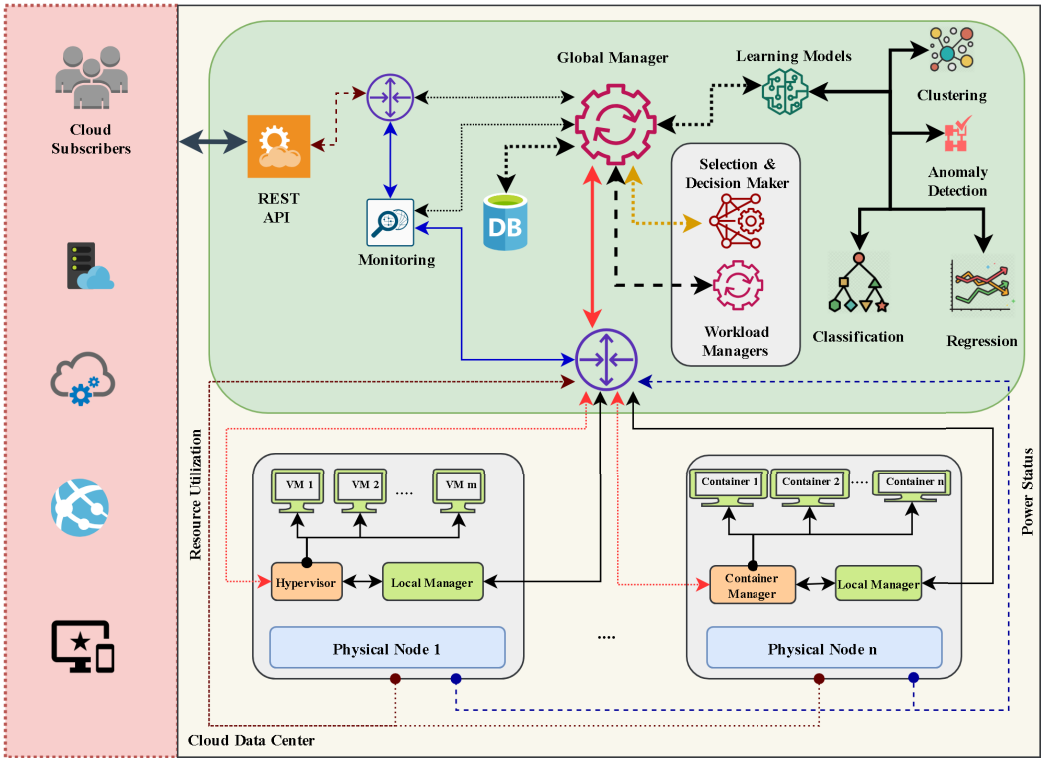

Figure 1.2: The proposed methodological framework.

center, incorporating the selection and decision-making subsystem. This subsystem works closely with the Global Manager to select the best-fitting resources for incoming workloads and to make strategic decisions about scaling, load balancing, and other operational aspects managed by local and workload managers.

(4) **Cloud Data Center Resources Utilization:** At the base of the framework lies the physical resources of the cloud data center. The VMs and containers are managed by Hypervisors and Container Managers, respectively. Local managers supervise and manage resource utilization at the node level, ensuring optimal performance and resource usage.

(5) **Feedback Loop:** A critical aspect of the framework is the feedback loop. Decisions and actions at the cloud data center layer are monitored and fed back into the system. This information helps to refine learning models and decision-making processes continuously.

This framework aims to create a self-optimizing, scalable, and efficient CRMS that can adapt to the dynamic needs of cloud subscribers while minimizing operational costs and maximizing resource utilization. Integrating advanced ML models with real-time monitoring and strategic decision-making actions is at the core of this innovative approach to cloud resource management.

1.6 Research Contributions

The research presented in this thesis delves into cutting-edge techniques that aim to improve workload segmentation and predictive modeling in cloud computing environments, offering valuable insights for cloud decision-making systems. The main contributions are outlined below.

1.6.1 Clustering Techniques for Cloud Workload Management

Within the scope of this research, we make a substantial contribution by creating a clustering methodology specifically tailored for dividing cloud workloads into segments. The proposed method considers various perspectives to form the clustering results influenced by the workload preparation process. It can adjust to ever-changing and complex cloud environments, verified by

utilizing actual cloud workloads. The main objective is to enhance monitoring performance and clustering quality within the data centers. The primary contributions are as follows.

(1) **Exploration of Clustering Techniques:** The study delves into various clustering methods, each with unique attributes, making them suitable for different data preprocessing contexts. The focus is on automating workload categorization within cloud data centers using clustering and advanced data preprocessing methods. The research underscores the critical role of advanced data preprocessing techniques, integrated seamlessly with clustering methods, to achieve precise workload segmentation considering various perspectives.

(2) **Advanced Ensemble Clustering Technique:** We propose an innovative ensemble clustering approach designed to meet the challenges of the dynamic nature of cloud workloads. This approach offers a clustering for cloud workloads supported by various segmentation schemes. The novelty of this technique involves combining various data preprocessing pipelines with different base-clustering models. The outcomes of the base-clustering pipelines are filtered and then fed into a meta-clustering model to produce the final clustering results.

(3) **Optimization of Clustering Model Selection:** To select the most effective base-clustering pipelines, we proposed a combined scoring mechanism based on the Silhouette score, Calinski-Harabasz index, and Davies-Bouldin index. This score selected the high clustering quality of various perspectives, improving the final workload segmentation efficacy.

This research contributes to cloud workload management by introducing advanced clustering techniques optimized for the unique challenges of cloud workloads. The focus on automated workload categorization, coupled with the strategic use of ensemble clustering and comprehensive evaluation methods, sets a new standard in cloud resource management and segmentation quality.

1.6.2 Single-Output Prediction for Cloud Computing

This research delves into single-output predictive modeling within cloud environments, presenting a series of methodological advancements and innovations to manage the cloud data center in various aspects. The exploration is structured into three key contributions, each focusing on a distinct aspect of predictive modeling within the cloud data centers, outlined as follows.

(1) **Advanced Multilevel Learning Model:** This study segment investigates a novel multilevel learning model to forecast CPU utilization in cloud data centers. The model stands out for its high prediction accuracy, achieved by integrating anomaly, clustering, and ensemble-based prediction models across three distinct learning stages. This advanced approach is pivotal for enhancing prediction precision in resource utilization within cloud infrastructure.

(2) **Proactive Cost Estimation Approach:** We introduce a proactive, regression-based predictive modeling strategy for cost estimation incorporating various target transformation methods to boost the prediction accuracy. This approach is particularly valuable for navigating the complexities of prediction-based pricing models in cloud services. It allows both cloud service providers and users to estimate costs more accurately, leading to more effective pricing strategies and financial planning.

(3) **Imbalance and Ensemble Learning for Load Prediction:** We proposed a novel prediction model incorporating both imbalance and ensemble learning techniques to refine load prediction accuracy. The objective is to enhance host load prediction precision within cloud computing systems, a crucial component for optimal resource management in ever-evolving cloud environments. This approach addresses the challenges of the imbalanced computing systems posed by fluctuating workloads and diverse resource demand patterns, ensuring more efficient and responsive cloud operations.

These contributions offer a comprehensive insight into the latest developments in predictive modeling within cloud data centers using the combination of standard ML techniques and various analysis methods. They showcase technological advances and highlight their practical applications, underlining their significance in improving operational management. Therefore, this research represents a significant step forward in the field of cloud computing, offering both theoretical and practical advancements in cloud resource management and operational efficiency.

1.6.3 Multi-Output Prediction in Serverless Computing

In the rapidly advancing field of serverless computing, the demand for effective predictive techniques for function invocation becomes crucial. This research study presents a comprehensive set

of innovations to enhance invocation predictability in serverless architectures at various levels. This approach enables the uncovering of detailed insights into the specific behaviors of workloads from different perspectives. The following are the key contributions.

(1) **Function Invocation Predictive Modeling:** We present a predictive modeling framework that employs multi-output regression models to predict the occurrence of function invocation at different levels of analysis. This framework is intended to forecast the upcoming function invocation patterns in serverless environments at the user, application, and function levels, resulting in a more accurate, generalized, and cost-effective prediction mechanism.

(2) **Advanced Windowing Technique Analysis:** We explore the impact of varying windowing configurations on the performance of predictive models for each level of analysis. Our comprehensive analysis of window sizes, step sizes, and target window sizes combining dimensionality reduction offers significant insights into their effects on the accuracy and computational load of multi-output regression models.

(3) **Dimensionality Reduction and Data Complexity:** We provide an investigation into dimensionality reduction using Principal Component Analysis (PCA). This analysis simulates the interaction between different window sliding configurations, examining the influence of dimensionality reduction on the data size complexities and the model's efficacy. We propose a strategic methodology for large-scale data that optimizes the trade-off between maintaining data goodness and enhancing computational efficiency.

(4) **Comparative Analysis Framework Development:** We establish a robust comparative analysis framework that evaluates an array of inherently multi-output regression models under different windowing configurations. Based on real workload data derived from the Azure Functions dataset, ensuring practical applicability, we evaluate the models' performance using multiple performance metrics to identify optimal model configurations.

Collectively, these contributions offer a thorough examination of the suitability of multi-output regression models that integrate diverse analysis techniques to streamline and forecast function invocation across different tiers in serverless computing despite their intricate characteristics.

1.7 Thesis Organization

The structure of this thesis is organized as follows: Chapter 2 delves into a comprehensive review of the existing literature, focusing on key components of cloud resource management, such as workload segmentation, predictive modeling techniques, and multi-output prediction strategies in serverless architectures. Chapter 3 explores advanced clustering methods in cloud workload management, detailing different techniques and their application in cloud environments, and introduces a new ensemble clustering approach. Chapter 4 is dedicated to single-output predictive modeling in cloud computing, discussing a multilevel learning model for CPU utilization, regression-based methods for cost estimation, and imbalance and ensemble learning for load prediction within cloud data centers. In Chapter 5, the focus shifts to multi-output predictive modeling for serverless computing, examining the use of multi-output regression models for function invocation prediction and the impact of various analysis methods on these models. The thesis concludes with Chapter 6, which synthesizes the research findings, outlines the contributions of this study, and proposes directions for future work in the field of cloud computing technologies.

Chapter 2

Background and Literature Review

This chapter presents an in-depth analysis of various methods used to manage cloud resources. In Section 2.1, we explore different approaches used for the segmentation process of cloud workloads. Section 2.2 examines predictive modeling techniques used in cloud resource management. Finally, in Section 2.3, we investigate multi-output prediction strategies in serverless architectures.

2.1 Segmentation Strategies in Cloud Workloads

Cloud services are offered in various categories, such as infrastructure, platform, or application-based, customized to meet users' distinct needs [12]. Given the limited resources boundary within the data centers, implementing effective resource management strategies is imperative to maximize provider profits and meet user demands [13]. However, significant emphasis has been placed on improving the efficiency and effectiveness of workload categorization in cloud computing systems [14]. This section reviews several prevalent techniques, outlines their strengths and limitations, and sets the stage for the innovative approach proposed in this study.

2.1.1 Conventional Clustering Methods

Conventional clustering techniques, such as KMeans and hierarchical clustering (with an early stop of tree construction), have been widely used for the cloud workload categorization process [15]–[24]. Implementing these methods offers a streamlined and efficient approach to handling

various data sets. Their simplicity provides a greater degree of manageability, providing a reliable means of data management that can prove invaluable in various professional settings. However, these clustering algorithms have limitations. KMeans requires the user to pre-determine the number of clusters and uses a random selection of initial centroids. Hierarchical clustering can be computationally expensive and requires the user to pre-determine the number of clusters. Early stopping reduces computational costs but may lead to suboptimal results [9].

Density-based clustering algorithms have been proposed to overcome some of the limitations of traditional clustering techniques, which can be helpful to cluster cloud workloads efficiently based on density [25]–[28]. Compared to more conventional algorithms, these are more versatile because they can detect clusters of arbitrary shapes and do not require a predefined number of clusters. However, high-dimensional data and clusters of varying densities present challenges [29].

Deep learning-based methods such as autoencoder networks have emerged, allowing the handling of high-dimensional data and the discovery of complex patterns [30]–[35]. However, these methods require large amounts of data for training and can be computationally intensive. Additionally, they often behave like a black box, offering trim insight into the underlying clustering process.

2.1.2 Advanced Ensemble Clustering

Strehl et al. [36] introduced three heuristics for the cluster ensemble. The Cluster-based Similarity Partitioning Algorithm (CSPA) establishes pairwise similarity among cluster objects to form a unified clustering. The HyperGraph Partitioning Algorithm (HGPA) approximates mutual information, turning the ensemble problem into a hypergraph partitioning task. The Meta-CLustering Algorithm (MCLA) focuses on identifying and merging groups of clusters by addressing cluster correspondence. Another significant work in ensemble clustering is by Topchy et al. [37], who introduced a graph-based methodology. The approach combines different clusterings in a hypergraph, where each hyper-edge represents a cluster. They then proposed a graph partitioning algorithm to find the consensus function. Meta-clustering approach has also been explored [38]–[41]. In this method, they treated each base clustering solution as an entity. They are subjected to another clustering round to identify analogous solutions, culminating in a consensus solution. Nevertheless, it is crucial to develop a method that is suitable for the diverse data sets and tasks, considering the

range of techniques at hand, as no single method can universally accommodate all data forms.

It is essential to note that even modern clustering techniques inevitably exhibit inherent limitations in categorizing cloud workloads. The inherent complexity and diverse characteristics of cloud workloads frequently challenge the suitability of traditional clustering algorithms. Moreover, the effectiveness observed with a specific data set rarely guarantees consistent performance across different data sets, emphasizing the need for more flexible and multifaceted techniques. The rapid advancement of technology has resulted in greater diversity and variability in cloud workloads, creating a need for robust and adaptable clustering methodologies. Ensemble clustering strategies are promising and productive methods that combine the benefits of multiple distinct group schemas. Combining different clustering results enhances categorization reliability and effectively addresses the limitations of individual approaches.

2.1.3 Summary

It is necessary to investigate the use of clustering techniques to categorize cloud workloads, particularly when combined with sophisticated data preprocessing pipelines. This investigation enables us to reveal the hidden structure within the data, resulting in multiple legitimate clustering schemes. Ensemble clustering is regarded as a promising approach in this context. We are motivated to conduct this research to bridge the existing gap by effectively integrating different clustering techniques and overcoming their limitations seamlessly. The efficacy of ensemble clustering is acknowledged, but its application in cloud workloads involving multiple data preprocessing pipelines has yet to be explored. This research aims to be at the forefront of this field, providing in-depth understanding and practical solutions that align with the complex nature of cloud workloads.

2.2 Predictive Modeling for Cloud Resource Management

The ability to predict and effectively manage system performance, resource utilization, and cost is paramount in cloud computing. The following sections present a detailed exploration of various cloud management methods, highlighting the pursuit of more efficient, reliable, and cost-effective solutions based on high prediction accuracy. We provide insights into challenges and advancements

of strategically predicting performance and managing resources in cloud environments.

2.2.1 Multilevel Learning-Based CPU Utilization Prediction

The imperative to predict CPU utilization accurately finds its roots in both performance optimization and efficient resource allocation within cloud computing systems. This section follows the development of CPU prediction models, highlighting how different ML methods have been used to address the difficulties associated with them effectively.

Historically, the initial approach to CPU utilization prediction was based primarily on time series analysis [42], [43]. Techniques such as AutoRegressive Integrated Moving Average (ARIMA) have been famous for their ability to model and forecast time series data [44]. Autoregressive models, which predict future data as a linear function of past values, found their place in early cloud environments. Exponential smoothing assigns exponentially decreasing weights to past observations and is commonly used for prediction [45]. The Holt-Winters method based on exponential smoothing is more flexible to capture trends and seasonality in data [46], [47]. While these models were adept at dealing with stable and repetitive patterns in CPU utilization, they struggled when faced with rapid and unpredictable fluctuations common in modern, dynamic cloud environments.

Anomaly detection identifies data points that deviate significantly from expected patterns, indicating potential issues or unusual occurrences. In CPU utilization, sudden spikes or drops can indicate system malfunctions or other unexpected events. Techniques such as the Isolation Forest [48], [49], One-Class Support-Vector Machine (SVM) [50], and the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [51], [52] have been widely employed to detect anomalies within resource utilization. The Isolation Forest algorithm has gained recognition for its effectiveness in identifying anomalies in datasets with a high-dimensional feature space, which is often found within cloud environments.

Using clustering techniques, we can group data points that exhibit similar attributes or patterns. This technique is instrumental in CPU prediction, as it aids in the identification and classification of different utilization patterns. Conventional clustering techniques, such as KMeans and hierarchical clustering, have been used for the resource usage categorization process [19], [53]. The KMeans clustering technique segments the data by detecting similarities in CPU utilization patterns based on

randomly initial centroids. On the other hand, hierarchical clustering offers a tree-based depiction of data groupings, enabling a more profound understanding of hierarchical patterns and dependencies on CPU utilization. Performing clustering as a preliminary step can enhance prediction accuracy by grouping similar data points with shared characteristics, thereby improving the subsequent learning process upon including the cluster information.

Regression models, which employ historical data analysis to predict continuous values, have typically served as the basis for forecasting CPU utilization. These foundational methods operate under the assumption of a linear correlation between input variables and output. However, with the increasing complexity and dynamic of cloud environments that lead to more complex usage patterns, non-linear regression models such as polynomial regression [54], [55], support vector regression [56], [57], and random forests [58], [59] have been explored by many researches. Recent trends have also seen the adoption of deep learning-based regression models, particularly neural networks. Recurrent Neural Network (RNN)s [60], [61] and Long Short-Term Memory (LSTM) networks [62], [63] have shown a potential to capture long-term dependencies in time series data, making them suitable for predicting various CPU utilization patterns in cloud data centers.

Developing techniques to forecast CPU utilization has moved from traditional time series models to more complex ML approaches. As cloud systems become increasingly complex and dynamic, the combination of anomaly detection, clustering, and regression models offers a promising way to address the multifaceted issues of CPU prediction. By combining these models, a comprehensive approach to CPU prediction can be achieved, which involves detecting anomalies, grouping similar data points, and creating regression models, considering the results of the segmentation models to make precise predictions. This approach allows cloud providers to optimize system performance and avoid expensive downtime. Furthermore, this strategy can be applied to other data analysis and prediction areas, making it a valuable asset in various fields.

2.2.2 Imbalance and Ensemble Methods in Cloud Load Prediction

In cloud environments, host load prediction is a helpful tool for system administrators to plan and optimize resource usage, preventing performance bottlenecks or failures. Researchers use various approaches to manage cloud data center infrastructure to reduce energy consumption and avoid

SLA violations while maximizing resource utilization.

Many approaches have been developed for performance prediction, ranging from simple statistical models and time-series analysis to more complex ML and Artificial Intelligence (AI) techniques [64]. Lu et al. [65] proposed a forecasting model that uses an improved version of the KMeans clustering algorithm and a Back-Propagation Neural Network (BPNN) to predict changes in workload in a cloud system. Gao et al. [66] proposed a Bidirectional Long-Short Term Memory (Bi-LSTM) recurrent neural network for forecasting task failures in a large-scale cloud data center. Jodayree et al. [67] provided a rule-based algorithm for predicting maximum workload in cloud computing systems to enhance resource management ability to assign workloads and avoid network overloading dynamically. They compare predicted workloads using historical data to assess whether an overload scenario occurs. However, biased outcomes may occur when evaluating overload scenarios in a prediction without considering the imbalanced state of the hosts.

For prediction models based on ensemble learning techniques, Kim et al. [68] introduced an ensemble workload prediction framework that combines the strengths of several ML predictors to accurately forecast workload characteristics in cloud data centers. The framework employs a multi-class method to dynamically assign weights to each predictor based on their accuracy over time. Feng et al. [69] proposed an ensemble model for workload prediction with adaptive sliding window and time locality integration. The adaptive sliding window algorithm improves accuracy and minimizes overhead by accounting for trend correlation, time correlation, and random workload fluctuations during online prediction. They also proposed an error-based integration strategy incorporating a time locality concept and a multi-class algorithm to combine the models.

Precisely predicting system performance can enhance efficiency and reliability, guaranteeing that users' needs are met promptly. Despite the widespread use of various techniques, they have significant limitations. Statistical techniques often make strong assumptions about the data, such as linearity and normality, which may not hold in dynamic cloud computing environments. Time-series analysis techniques can capture complex temporal patterns. However, fitting a separate model for each host in the cloud data center can be computationally expensive.

In addition, ML techniques typically assume balanced data. However, in practice, the data distribution for physical machines' overload, underload, and normal load states is often imbalanced,

with one state occurring more frequently than others. This imbalance can lead to biased models towards the more common state, reducing their accuracy for predicting less common states. Given these limitations, there is a clear need for better predictive models incorporating advanced methods to forecast load states accurately in cloud computing systems. Advanced ML techniques, such as imbalance and ensemble learning, offer promising alternatives for overcoming these challenges.

2.2.3 Summary

The literature highlights a variety of methodologies and strategies employed to predict system performance, anticipate workload changes, and optimize cost efficiency within cloud environments. These include using statistical models and ML techniques, each catering to different aspects of cloud computing challenges. The reviewed studies highlight the need for continuous refinement, particularly in handling dynamic cloud computing workloads. The literature delves into the prediction of CPU utilization, a critical component of performance optimization. Traditional time series models have evolved into more sophisticated ML approaches to address the dynamic nature of cloud systems. Incorporating anomaly detection and clustering methods with the prediction models provides a robust framework for understanding and predicting CPU usage patterns. Also, when designing approaches that can cater to multiple hosts, it is crucial to consider the generalization problem. With this consideration, we can ensure that the approaches developed are reliable and effective across various platforms to achieve desirable outcomes.

In addition, using ensemble and imbalance methods in cloud load prediction is very important. These advanced techniques are presented as solutions to overcome the limitations of traditional predictive models, especially in producing highly accurate predictions while handling imbalanced data distributions and rapidly changing load states. Integrating multiple predictors and employing strategies such as multilevel learning, sliding windows, and ensemble models can be a desired solution to improve the accuracy, generality, and adaptability in performance prediction of the running deployments and hosts within the cloud data centers.

2.3 Multi-Output Prediction in Serverless Computing

This section reviews the existing research studies on function invocation within serverless computing environments, focusing on predictive analysis. This review seeks to investigate the complexities of serverless computing, focusing on the necessity and potential of multi-output prediction models. This exploration is essential in order to comprehend and improve the effectiveness and accuracy of function invocation at multilevel pattern architecture, a key element of serverless computing that needs to be sufficiently examined in this rapidly advancing area.

2.3.1 Overview of Serverless Computing

Serverless computing is an innovative approach in cloud computing that involves a significant change in the way architectures are designed, moving away from traditional server-focused models and instead abstracting server management. The evolution of this computing concept has been extensively documented, demonstrating the progression from an emerging idea to a widely accepted cloud service model [70], [71].

Serverless computing is characterized by its ability to allocate resources on-demand and unique billing models, setting it apart from traditional cloud services. Serverless architectures dynamically allocate resources based on specific function calls or events. This process guarantees that users are only charged for the resources that are actually utilized. This strategy provides both cost advantages and is well-suited to the flexible demand of cloud-based applications, enhancing the power and energy efficiency of cloud data centers [70], [72]. The act of calling functions in serverless computing is fundamental to its operational framework. Functions, which are self-contained units of application logic, are executed in response to various events or triggers. This model promotes a flexible and responsive environment, making serverless computing highly suitable for event-driven applications [73], [74]. The effectiveness and reliability of these calls are essential, as they directly influence the efficiency and cost-efficiency of the serverless infrastructure [75], [76].

Furthermore, the rise of AI and ML has introduced new dimensions to serverless computing. Advanced techniques such as AI-based resource allocation and adaptive auto-scaling are being integrated to optimize function invocation and resource management. This phenomenon signifies the

continuous advancement and growing complexity of serverless architectures [77].

Serverless computing has emerged as a revolutionary aspect of cloud computing because of its ability to allocate resources on demand, its cost-efficient billing models, and its focus on executing functions. The ongoing evolution of this framework, characterized by progress in AI and ML and its increasing compatibility with contemporary computing needs, underscores its expanding significance in the field of cloud computing. Function invocation is essential in this model, as it forms the basis for the efficient operation and economic sustainability of serverless architectures.

2.3.2 Function Invocation in Serverless Computing

Invoking functions in serverless architecture is fundamental for applications to respond to different events, representing a shift from conventional server-based methods. This section examines the mechanics of function invocation, examines research that concentrates on optimization and discusses the consequences of invocation patterns on resource management and system performance.

In a serverless environment, functions are invoked in a manner that is driven by events. These functions are executed in response to specific triggers, such as HTTP requests, file uploads, or other events that occur in the cloud. These invocations follow a model where the cloud provider dynamically handles the allocation and scaling of the underlying infrastructure. The execution of serverless applications, encompassing both cold and warm starts, has a substantial impact on their performance and responsiveness [74], [78]. The optimization of the invocation process primarily focuses on minimizing latency and resolving cold start problems. Cold starts happen when a function is called after being inactive, necessitating the cloud provider to assign resources before execution. This delay can negatively impact the user experience. Studies have investigated different approaches to reduce these delays, including examining tail latency in serverless clouds [79], comprehending the variability in function invocation times [80], and creating techniques to reduces the service time and enhance the warming of serverless functions [81].

The manner in which functions are called has direct consequences for the management of resources and the overall performance of the system. Optimal invocation prediction strategies can result in enhanced resource allocation, reduced expenses, and enhanced user satisfaction. The difficulty lies in precisely forecasting the patterns of invocation and adjusting the allocation of resources

accordingly. The studies reviewed offer valuable insights into the optimization of serverless computing for handling diverse workloads, thereby ensuring optimal performance and efficient utilization of cloud resources [74], [78].

2.3.3 Use of Cloud Workload Traces for Predictive Analysis

The utilization of cloud workload trace has emerged as an essential tool in serverless computing for predictive analysis, facilitating a more sophisticated comprehension of serverless environments. This section provides an overview of important studies that have utilized different types of evidence, examining their research methods, results, and the wider significance of using real-world data to forecast serverless computing workloads.

Cloud workload traces, which are essentially logs of past server utilization and user request patterns, offer invaluable data for simulating and analyzing serverless computing environments. These traces provide valuable information about common usage patterns, resource demands, and performance limitations, which are essential for optimizing serverless infrastructures [80], [82].

Cloud workload tracers have been utilized in numerous studies to conduct predictive analysis in the field of serverless computing. The FaaS-sim framework utilizes workload traces obtained from real-world testbeds to simulate execution time and resource utilization on various computing hardware. Similarly, the research presented in [83] employs cloud workload traces to predict serverless computing workloads using advanced ML techniques like the Wasserstein Adversarial Transformer (WAT). Another study addressing the mitigation of the cold start problem in serverless computing [84] analyzes sudden spikes in workload traces and their impact on service delivery.

Utilizing cloud workload traces for predictive analysis provides numerous advantages. It allows for a more precise evaluation of resource requirements and user demand patterns, which aids in improved capacity planning and resource distribution. This approach also aids in the identification and resolution of performance issues, such as latency and cold starts, by offering a practical framework for simulation and testing [85], [86]. Nevertheless, there are constraints to this methodology. Examining real-world workload traces at a specific pattern architecture level may occasionally provide a reliable representation of load patterns, but this is limited by the ever-changing nature of user behavior and technological progress. Furthermore, the utilization of real user data gives rise to

concerns regarding privacy and data security [87].

The use of cloud workload traces in the development of serverless computing operational management offers a practical approach for performing predictive analysis, thereby significantly enhancing the optimization of serverless platforms. Incorporating actual data from the real world into the development of serverless computing prediction models is extremely beneficial for enhancing efficiency and responsiveness despite the specific difficulties it may present.

2.3.4 Multi-Output Prediction Models

In serverless computing, where dynamic resource allocation and efficient service delivery are paramount, multi-output prediction models emerge as a critical tool for managing complex workload patterns. This section examines multi-output prediction models, their significance in serverless computing, and their potential to improve the invocation prediction process.

Multi-output prediction models are sophisticated analytical instruments that have the ability to predict multiple dependent variables or outputs simultaneously. Within the realm of serverless computing, these models hold significant importance as they possess the capability to examine and forecast various aspects of a function invocation, including invocation frequency, execution duration, and resource utilization. These factors are essential for the enhancement of serverless platforms [88]. By utilizing these models, serverless computing systems can attain enhanced resource provisioning accuracy, reduce latency, and enhance overall service performance.

The literature reveals a growing interest in employing multi-output prediction models in cloud computing and related fields. For example, the study by Liu and Xu utilizes a Multi-output Support Vector Regression (MSVR) model combined with a Immune Clone Selection Algorithm (ICSA) to improve big data in cloud computing platforms [89]. This approach highlights the capability of multi-output prediction models in dealing with intricate, high-dimensional data that is frequently encountered in cloud environments. Another noteworthy contribution is the TPPFaaS framework, which models serverless function invocations using Temporal Point Processes (TPPs), providing insights into workload prediction in serverless computing [88].

Serverless computing enables multi-output prediction models that can simultaneously predict multiple critical aspects of function invocation. These models can forecast invocation frequency,

helping to anticipate demand spikes and scaling resources accordingly. In addition, they can approximate the time it takes for a task to be completed, which is crucial for effectively managing time-sensitive tasks and minimizing the occurrence of cold starts. Also, they can forecast resource utilization patterns, which improve computational resources and optimization of costs. The literature review showcases the adaptability and efficacy of these models, emphasizing their increasing significance in the changing realm of cloud computing.

2.3.5 Summary

Although there is an increasing amount of research on serverless computing, there are still areas for improvement, especially when it comes to multilevel predictive analysis. Contemporary research, despite being thorough, typically requires a multifaceted methodology for predictive analysis. Research primarily examines particular aspects of serverless environments, such as cost or performance, without effectively incorporating these factors into a comprehensive predictive framework. There is a significant gap in current research regarding the need for comprehensive models that can address multiple aspects of serverless computing. These aspects include continuous invocation frequency at the application level for resource allocation, cost estimation at the user level, and execution timing at the function level.

The current predictive models frequently encounter difficulties in accurately predicting serverless computing workloads because of the extremely dynamic and event-driven characteristics of serverless computing. These models have to include the inherent variability and unpredictability present in the time series of function invocation trace. The difficulty lies in developing models that possess sufficient flexibility and generalizability to effectively manage diverse event-driven functions across different time intervals within serverless environments.

There is a distinct requirement for more sophisticated and subtle predictive models that can accommodate the ever-changing characteristics of serverless computing. These models need to have the ability to perform multilevel analysis, forecast diverse outcomes, and adjust to dynamic conditions in real time. Such predictive models allow serverless computing to effectively adapt to the changing requirements of contemporary cloud-based applications and services.

Chapter 3

Advanced Clustering Techniques in Cloud Workload Management

In this chapter, we delve into the sophisticated realm of clustering techniques, focusing on the segmentation of cloud workloads. The effective segmentation of workloads in cloud environments is crucial for optimizing performance, cost efficiency, and scalability. Initially, we explore various clustering techniques for workload segmentation, highlighting their unique attributes and suitability in different data pipelines (Section 3.1). Then, we explore an advanced concept of ensemble clustering (Section 3.2). This section explores how combining multiple clustering approaches can provide a more comprehensive and nuanced analysis of cloud workloads. This innovative strategy is particularly pertinent to address the multifaceted challenges posed by the dynamic nature of cloud environments, ensuring more efficient and effective workload management.

3.1 Dynamic Workload Segmentation based on Multiple Data Pipelines

Integrating ML technology into CRMSs marks a significant step for cloud providers to understand and address customer needs, thus refining their decision-making processes. This advancement is crucial for streamlining business operations and increasing customer satisfaction. Central to the development of effective ML models is the preprocessing of observed data, a stage that critically shapes the learning process and the accuracy of predictions [8]. This preprocessing involves several

key tasks: data cleaning, integration, transformation, and reduction. Each of these tasks is geared towards enhancing the data's quality and utility for ML model development.

In the realm of cloud computing, workloads are characterized by their high variability and the presence of unforeseen patterns, presenting unique challenges in knowledge discovery. This complexity necessitates the deployment of robust data preparation pipelines. Properly executed, these pipelines not only improve the quality of the data but also facilitate the extraction of valuable insights. When data is meticulously preprocessed, the resulting ML models are not only more precise but also more reliable. These enhanced models can then be seamlessly integrated into the Selection and Decision-Making (SDM) modules of CRMSs. This integration is crucial for achieving efficient and effective resource management in cloud data centers, ensuring that resources are optimally allocated to meet diverse and dynamic workload demands.

Categorizing cloud workloads is vital for improving monitoring procedures and managing cloud resources. This process can help allocate resources more effectively, reduce waste, and prioritize critical workloads and their requirements [23], [90]. This study introduces a methodology for categorizing cloud workloads by employing clustering techniques and multiple data pipelines. We emphasize the importance of employing precise data preprocessing techniques to guarantee precise and reliable clustering results based on different perspectives.

3.1.1 Workload Clustering Framework

The proposed framework utilizes clustering methods to partition the workload into subsets with similar characteristics to estimate performance metrics based on different data views. Different criteria are used to evaluate the clustering quality and determine the optimal number of clusters for the partitioning process within the cloud data centers. The labels produced by the clustering process serve as unique identifiers for distinct sets of tasks that share commonalities. The cloud provider can then use the cluster estimations to calibrate the task scheduling and placement algorithms to dispatch jobs into logical Virtual Computing Instance (VCI)s such as VMs and containers. Therefore, by providing valuable insights into the characteristics and behavior of cloud workloads, it enables more efficient and effective management of cloud resources.

3.1.1.1 Typical System Design

The typical CRMS comprises global, workload, and local agent managers, distributed throughout the data center's higher and lower logical layers. The global manager resides in a controller node at a higher layer, connected to workload managers and all distributed local managers at the lower layer. Each physical node is connected to a local manager that manages the node's resources through VCI managers (Virtual Machine Manager (VMM) and Container Manager (CM)). They oversee to collaborate with the associated global manager to provision resources and accommodate the workload requirements. However, the CRMS utilizes task delivery modules working together to complete user requests on properly organized resources that adhere to SLA requirements. The following additional information pertains to each module:

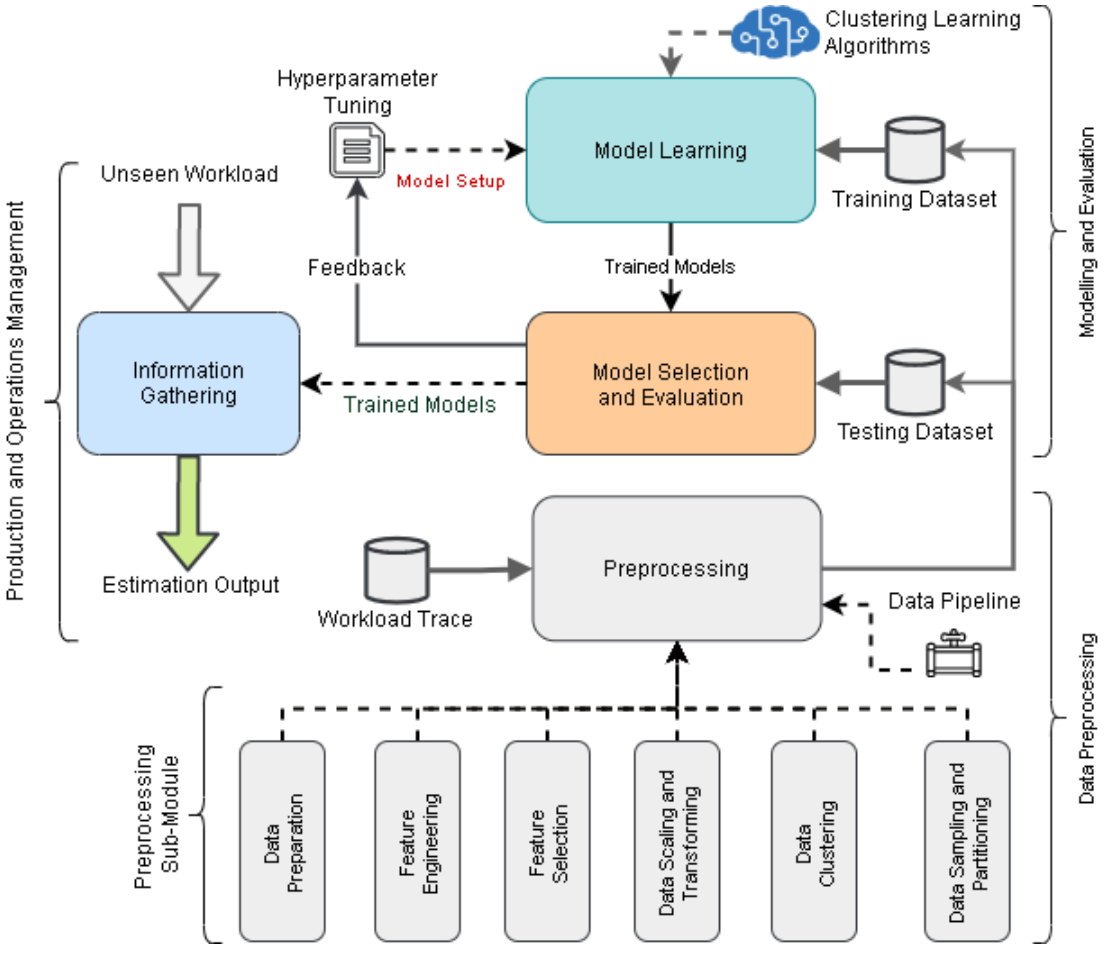

Figure 3.1: End-to-End machine learning development lifecycle workflow.

Monitoring Module: Within data centers, the monitoring module is a crucial component designed to assist cloud managers in keeping track of the workload and infrastructure resources. It allows managers to keep a close eye on the performance of various systems and helps them make informed decisions about resource allocation and utilization. This module collects user request requirements, resource capabilities, VCI utilization, and the status of physical nodes. The collected data is then stored in a data storage facility accessible by all data center managers. The monitoring module also coordinates with the learning module to preprocess the collected data. The data passes through given data pipelines in a predetermined order to prepare it for clustering. The output samples are then used to train the desired clustering models based on a predetermined model and configuration list. With the help of the monitoring module, cloud managers can ensure that their systems are functioning optimally and provide high-quality segmentation.

Learning Module: This module comprises the data preparation and the ML development and evaluation phases, as shown in Figure 3.1. The data preparation phase applies different data preprocessing and feature engineering techniques to form data pipelines and their configurations. These pipelines help transform the data into a suitable format for training and evaluating ML models. In the ML development and evaluation phase, the transformed data is used to train, assess, and hyper-parameter tune predetermined clustering models. The global manager utilizes the clustering outcomes to form actions like VCI migration maps through the SDM module and feeds the workload and local managers accordingly to optimize the cloud resource. Nonetheless, this study focuses on this module, which is discussed in detail in Section 3.1.1.2.

Selection and Decision-Making Module: This module manages the resources of the data center following the characteristics of the workload and the required resource demands by forming the operational management of CRMS. It plays a significant role in keeping the dynamic workload scheduling and consolidation process running smoothly. It involves cluster identification, a data-driven problem of identifying VCI configuration that conform to task-defined usage patterns and access policies with minimal overhead. Also, it coordinates with the information-gathering module that provides the expected future load demands and the required resource capacities of active VCI, including the current state of the physical resources. The SDM module uses this information to generate scaling and migration maps that optimize resource allocation and workload scheduling.

3.1.1.2 Clustering Workflow Pipelines

The workload clustering model classifies given tasks into k groups based on their characteristics such as resource utilization and execution costs. Two algorithms are proposed to create workflow clustering pipelines. These algorithms enable the learning module to preprocess the traces in various formats and build clustering models accordingly. The details of each algorithm are as follows.

Data Preprocessing and Feature Engineering: Data pipelines are a sequence of organized methods used to preprocess and engineer features in a prescribed manner. They implement procedures configured based on predetermined setups $PC = \{PC_1, PC_2, PC_3, ..., PC_d\}$, where d represents the number of data pipelines, considering various transformation layouts.

Algorithm 3.1 shows the preprocessing workflow. It starts by extracting a subset of the trace based on a given window size (W). This parameter determines the duration of the trace to be analyzed and used in the clustering process. This step is helpful when workload patterns change over time, as it allows us to capture and analyze the variations separately and form actions accordingly. Next, it extracts features based on resource usage and date and time indexes. The first step involves

Algorithm 3.1 Data Preprocessing and Feature Engineering

Input: $Tr \leftarrow SchedulingTrace, PC \leftarrow Config$
Output: CP $\{DataPipelinesWorkflow\}$

1: $CP \leftarrow [\,]$
2: **for** PC_i in PC **do**
3: $P.Sampling(PC_i[W])$
4: $P.FeatureExtractionBasedOnResourceUsage()$
5: $P.ExtractDateTimeFeatures()$
6: $P.OrdinalEncoder()$
7: $P.OneHotEncoder()$
8: $P.CounterEncoder()$
9: **if** $PC_i[Polynomial] == True$ **then**
10: $P.ProducePolynomialFeatures(PC_i[D])$
11: **end if**
12: $P.FeatureSelection(VT, MT)$
13: **if** $PC_i[ReduceDimensionality] == True$ **then**
14: $P.Reduction(PC_i[\text{COMPONENT}])$
15: **end if**
16: $P.FeaturesTransformation()$
17: $P.FeaturesScaling()$
18: $CP.Append(P.Fit(Tr, PC_i))$
19: **end forreturn** CP

transforming variables from the trace data that capture the resource utilization patterns of the tasks. These variables may include CPU utilization, memory utilization, disk I/O, network utilization, and other pertinent metrics. This step aims to identify the most informative variables that can differentiate between various task types and provide insight into the resource demands.

The second step involves obtaining essential information from the task timestamps. This information contains indexes such as year, month, day, hour, minute, and weekday, which indicate when each task was initiated and completed. By extracting these indexes, we can gain insight into how the workload fluctuates over time and how the clustering outcomes may be influenced by external factors such as seasonality and time of day and adapt resource management decisions accordingly.

The categorical variables are then converted into a numerical form that can be used to build ML algorithms. Categorical data is any information that consists of non-numerical values, such as subscription index, deployment index, VCI type, CPU core bucket, and memory bucket. In the proposed algorithm, three types of encoders are considered. 1) Ordinal encoding is a process for assigning a distinct integer to each category, such as CPU core bucket and memory bucket, ordered based on size. 2) One-hot encoding is a technique that produces a new binary variable for each category, such as VCI type. 3) Counter-encoding is a technique used to encode categorical variables by replacing each category, such as subscription and deployment indexes, with a count of its appearances in the data set while preserving the frequency-based ordering of the categories.

Subsequently, a polynomial function is utilized to produce new attributes by amalgamating existing numerical features, thus facilitating the capture of non-linear relationships between them. The degree of polynomiality (D) determines the degree of transformation. It is essential to exercise caution when selecting D, as increasing it produces excessive polynomial features that can lead to overfitting. This consideration highlights the essentiality of thoughtful decision-making when selecting the degree to be included in the model. Then, feature selection methods are applied to ensure that only informative variables are retained by removing less important ones. The low variance variables are less critical elements that can be removed according to a given threshold (VT). Pearson correlation can determine the correlation between the features within the given data set. Then, we can identify the features with multicollinearity, a statistical phenomenon representing two or more highly correlated variables that can be dropped according to a given threshold (MT).

Algorithm 3.2 Workload Clustering Workflow

Input: $Tr \leftarrow SchedulingTrace, W \leftarrow WindowSize$
Output: TCM $\{TrainedClusteringModels\}$

$PC \leftarrow GetPipelinesConfig(W)$
$CP \leftarrow GetDataPipelinesWorkflow(Tr, PC)$
$TCM \leftarrow [\,]$
for CP_i in CP **do**
    $X_{train}, X_{test} \leftarrow DataPreprocessing(T, CP_i)$
    $M, MC \leftarrow GetClusteringModels\&Configs()$
    for M_i, MC_i in $zip(M, MC)$ **do**
        $M_i.Train(X_{train}, MC_i)$
        $F_i \leftarrow M_i.Evaluate(X_{train}, X_{test})$
        if Tunable(M_i) **then**
            $TMC_i \leftarrow Tune(X_{train}, X_{test}, M_i, MC_i, F_i)$
            $M_i.Update(TMC_i)$
        end if
        $TCM.Append(M_i)$
    end for
end forreturn TCM

After that, PCA is employed to reduce the dimensionality of complex data sets while maintaining the most significant components that effectively capture the highest degree of variation in the original data. This approach selects appropriate components based on an explained variance threshold determined using a knee method proposed in [91]. This process can also help improve ML models' performance by reducing the amount of noise and irrelevant information in the data.

Finally, feature transformation and scaling techniques are utilized to normalize the produced data features to have similar scales and ranges. In feature transformation, a mathematical function is applied to the features to transform them into a new space, whereas, in feature scaling, the features are rescaled using a standard scaler method. This process is vital in ML because some algorithms are sensitive to the scale of the input features.

Workload Clustering Workflow: Multiple models for workload clustering are created for each data pipeline in this module. Algorithm 3.2 shows the steps of the clustering process, including the evaluation and tuning processes. The scheduling trace Tr is used as input data, filtered using a window size W. Then, fetch the planned list of data pipelines CP based on given setups PC. The workload manager forms the configuration policies of each data pipeline workflow. For each CP_i, data samples are produced by implementing a batch preprocessing stage that handles imputations

and feature extraction, using Algorithm 3.1. Afterward, the clustering models undergo training, evaluation, and tuning through a predetermined set of models M and their corresponding parameter setups MC. The superior models are then recorded and made available for utilization by the data center managers. The information-gathering module groups the workload into segments based on the best model clustering outcome. It then constructs estimations for each data segment using descriptive statistics or ML models such as regression or classification. Based on these estimations, the SDM module generates a VCI scaling and migration map. This map is then used to schedule, balance, and consolidate the active workloads.

3.1.2 Experimental Setup and Evaluation

The proposed model is evaluated using real cloud workloads, three clustering algorithms, and four distinct data pipeline configurations. Various performance metrics are used to examine the ideal number of clusters and the quality of the segmentation. More details are outlined below.

3.1.2.1 Clustering Evaluation Metrics

Clustering performance is crucial, as data clusters are often examined manually and qualitatively to determine their significance. When the ground truth label is unknown, the performance of the clustering method can be examined utilizing different intrinsic metrics. In this study, we used the most popular metrics, which are described as follows.

Silhouette Coefficient: The Silhouette Coefficient (SC) score is a metric for evaluating the clustering quality using the maximum internal coherence and separation of clustering outcomes [92]. This score concisely illustrates how clustering is performed within each cluster and between neighboring clusters. The SC score varies from -1 to 1, allowing us to evaluate the consistency within data clusters. The mean score can also show the overall clustering performance. The higher score indicates that the cluster data points are farther away from the neighboring points, while a negative score means that the cluster data points might be in the wrong group. A cluster with a score of zero or close to it indicates that the data points in the cluster are close to the boundaries and may overlap with other clusters. The SC score is calculated using the following formula.

$$SC = \frac{b - a}{max\,(a, b)} \tag{1}$$

where a is the mean intra-cluster distance (*mean distance between a data point and all others within a cluster*), and b is the mean nearest-cluster distance (*mean distance between a data point and all other ones in other clusters*). Euclidean distance can be used to calculate the mean distance. Generally, for two points p and q given by Cartesian coordinates in n-dimensional Euclidean space, the distance d can be calculated as the following equation.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2}$$

Calinski-Harabasz Index: The Calinski-Harabasz Index (CHI), also known as the Variance Ratio Criterion, can be used to evaluate the clustering outcomes [93]. It is a metric that estimates the dispersion ratio within a cluster and the mean dispersion between clusters. The dispersion can be determined by summing the squared distances. The higher value of CHI indicates a better-defined clustering performance that includes dense and well-separation groups. The CHI can be easily calculated as the following equation.

$$CHI = \frac{\mathrm{tr}\,(B_k)}{\mathrm{tr}\,(W_k)} \times \frac{n_E - k}{k - 1} \tag{3}$$

where n_E is the size of a sample data set E that has been clustered into k clusters, $\mathrm{tr}\,(B_k)$ is the trace matrix of the dispersion between clusters, and $\mathrm{tr}\,(W_k)$ is the trace matrix of the dispersion within the cluster, defined as follows:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)\,(x - c_q)^T \tag{4}$$

$$B_k = \sum_{q=1}^{k} n_q\,(c_q - c_E)\,(c_q - c_E)^T \tag{5}$$

where C_q refers to a group of data points within the cluster labeled as q. The variable n_q represents the total number of data points within cluster q, while c_q denotes the center of the same cluster. c_E

refers to the center of the sample data set, while the symbol T denotes the transpose of the matrix.

Davies-Bouldin Index: The Davies-Bouldin Index (DBI) is another evaluation metric for clustering algorithms. It measures the similarity between two clusters by calculating the ratio of within-cluster distances to the distance between the other clusters [94]. The minimum value of DBI is zero, and the model with a lower value indicates further apart and less scattered within the data, representing better clustering performance. The DBI can be computed using the following formula.

$$DBI = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} \tag{6}$$

where k is the total number of clusters, R_{ij} is a measure that represents how good the clustering scheme is between i^{th} and j^{th} clusters, calculated as follows.

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{7}$$

where s represents the average distance between each data point of a cluster and its centroid value calculated for both i^{th} and j^{th} clusters, d_{ij} is the distance between cluster centroids i and j. Small s and large d_{ij} give a minimum DBI value that indicates the best partitioning outcomes.

3.1.2.2 Microsoft Azure Public Workload

Microsoft Azure is a cloud computing platform that offers various services, including computing, storage, analytics, and networking, enabling businesses to build, deploy, and manage applications and services globally [95]. Cloud users may submit multiple jobs to any regional data center using single or multiple subscriptions. Each task operates on a VM within a deployment that provides the required resources for each task to run efficiently, considering its particular requirements. In our experiments, we used a random data set of 10000 VMs (running for one month) derived from the Azure Public Data set v2 [96]. The trace includes subscriptions and deployments indexes for each VM described as nominal categories. It also includes ordinal categories to show the CPU core and memory buckets of the VM and a nominal category representing the VM type. It also contains timestamps for starting and stopping VM, maximum and average CPU utilization, and the 95^{th} percentile of maximum CPU utilization as numerical features.

3.1.2.3 Pipeline Setups and Evaluation Results

Scheduling trace with a window size (W) of 30 days is subjected to various data pipeline setups, each employing a unique combination of preprocessing and feature engineering techniques in preparation for clustering. These setups may differ in the parameters used or the inclusion/exclusion of specific data preparation methods for each clustering technique. The results of each configuration are then compared to determine which configuration and methods perform the best in terms of the clustering quality evaluation metrics. With this flexibility, the best settings for each clustering method can be chosen and applied to the data for optimal clustering. However, we evaluate the proposed data pipeline configurations using three clustering algorithms (KMeans [97], Agglomerative [98], and MeanShift [99]) based on the implementation of sklearn [9]. The chosen clustering algorithms offer a diverse range of techniques for different data characteristics and use cases. KMeans provides a scalable, efficient centroid-based clustering. Hierarchical, particularly the Agglomerative method, offers a tree-structured approach for interpreting data hierarchies. MeanShift complements these by offering a density-based clustering ideal for complex data shapes and varying densities. Together, these methods form a comprehensive toolkit for addressing various clustering challenges.

Choosing the optimal number of clusters is among the most challenging aspects of the clustering process. The KMeans and hierarchical clustering algorithms require the number of clusters to be defined to group the data points accordingly. The distortion score can be used to choose the optimal number of clusters for each data pipeline. This score calculates the sum of the squared distances between each data point and its cluster centroid. The lower the score value, the better the clustering performance. However, by plotting the score for different clusters, we can choose the number of clusters that results in the most significant decrease in the distortion score. This spot is referred to as the elbow point because it is typically where the plot begins to flatten. However, mathematically, the distortion score can be expressed as:

$$D = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{8}$$

where k is the number of clusters, C_i is the set of data points assigned to cluster i, x is a data point, and μ_i is the centroid of cluster i. The objective of the clustering process is to reduce this score

to the greatest extent possible. In this study, using this score, we employed a knee point detection algorithm proposed in [91] to automatically detect the elbow point that corresponds to the optimal value of k, ensuring a reliable and objective analysis.

As previously mentioned, the parameters utilized or the inclusion/exclusion of specific methods for each clustering technique may vary across data pipeline configurations. By experimenting with different configurations, we can evaluate their effectiveness and select the optimal configuration for each clustering method. Detailed descriptions of the utilized pipeline configurations, including the evaluation results and comparison, are provided as follows.

Standard Pipeline (SP): The data pipeline is a systematic approach to preparing data for clustering, and it involves a series of steps to ensure that the data is in a suitable format and good quality. The pipeline includes several preprocessing techniques, such as sampling, feature extraction based on resource usage, and creation and deletion timestamps of VCI. One of the principal steps in the pipeline is encoding categorical variables, which involves converting categorical data into a numerical format that the learning process can understand. This step is essential because many ML algorithms can only work with numerical data. Another important step is removing features with zero variance and those with perfect collinearity ($VT = 0$ and $MT = 0.99$). Features with perfect collinearity provide identical information, and their inclusion in the model does not add value to the learning process. Zero variance features have the same value for every data point in a data set, and their elimination enhances model performance and decreases computational complexity. After removing redundant features, the pipeline performs feature transformation and scaling. Feature transformation involves converting the data into a more suitable format for analysis. Scaling involves standardizing the data to a standard scale to prevent any one feature from dominating the analysis. By following these steps, the data pipeline ensures that the data is of high quality and sets the stage for an accurate and reliable data clustering process.

After that, we used the quantile transformation method to transform the features into a uniform distribution using sklearn implementation [9]. This method transforms skewed data into a normal distribution by calculating the quantiles of the initial data and mapping them to the quantiles of the normal distribution. We also utilized the min-max normalization method to scale numerical data in a fixed range between 0 and 1. It is applied to change all the features of the data set to the same

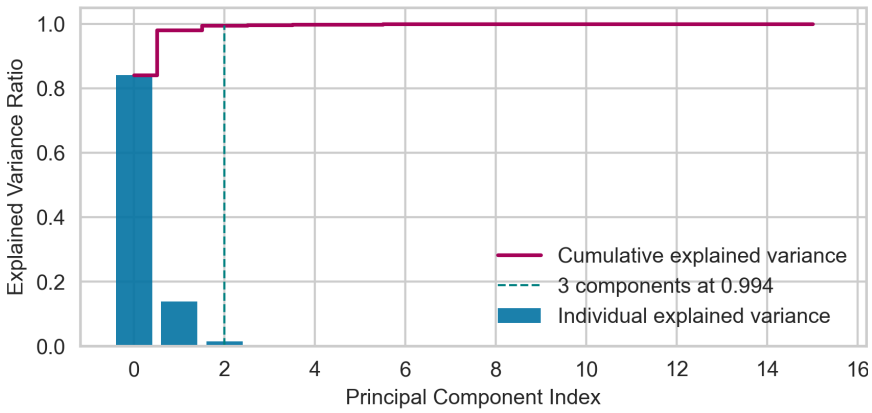

Figure 3.2: Explained variance plot for the transformed data using Standard Pipeline (SP), excluding the normalization and transformation methods.

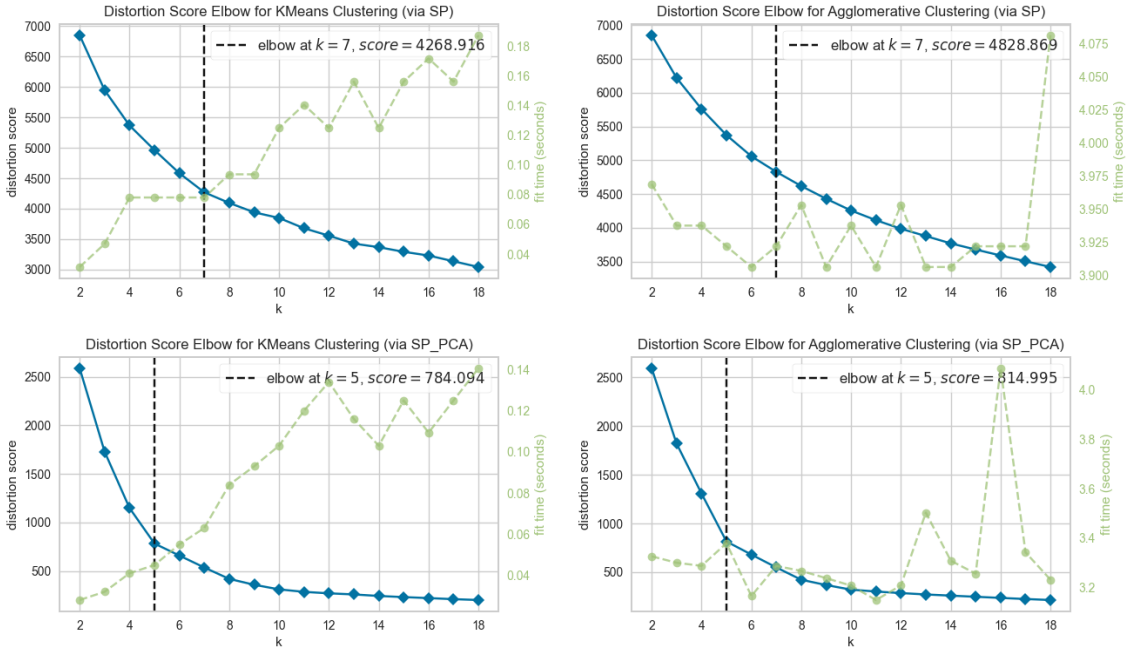

Figure 3.3: KElbow plots with distortion scores and training time for KMeans and Agglomerative clustering methods utilizing the Standard Pipeline (SP), with and without the application of PCA.

scale. The formula to perform min-max normalization on a feature X is:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{9}$$

where X_{min} and X_{max} represent the minimum and maximum values for the given feature X.

Table 3.1: Comparison of Clustering Methods Utilizing the Standard Pipeline (SP), with and without the application of PCA.

Metrics	KMeans	Agglomerative	MeanShift	Pipeline
CHI	14532.47	13886.72	5856.82	SP_PCA
	2090.31	1654.79	1377.47	SP
DBI	0.62	0.66	0.70	SP_PCA
	1.52	1.93	1.09	SP
SC	0.56	0.55	0.52	SP_PCA
	0.21	0.13	0.44	SP

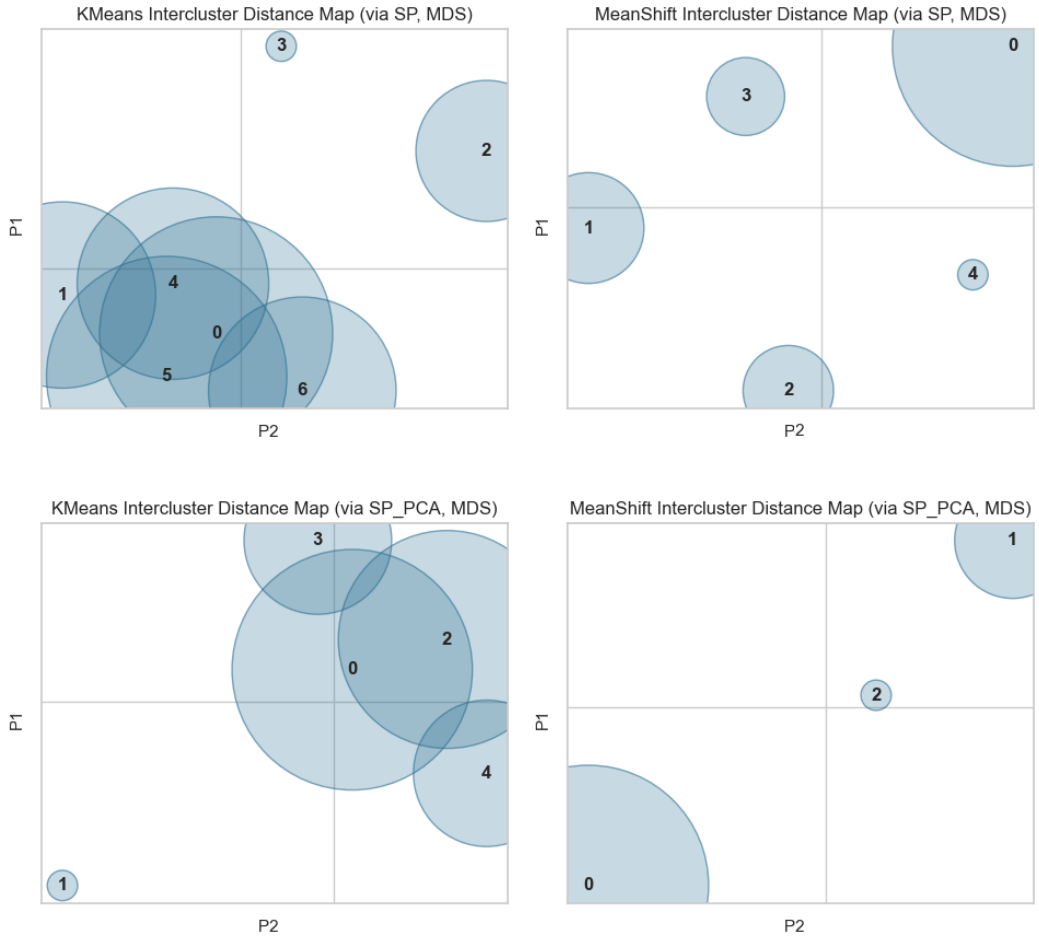

Figure 3.4: Intercluster distance map for KMeans and MeanShift clustering methods using the Standard Pipeline (SP), with and without the application of PCA embedded via the MDS into two features (P1, P2).

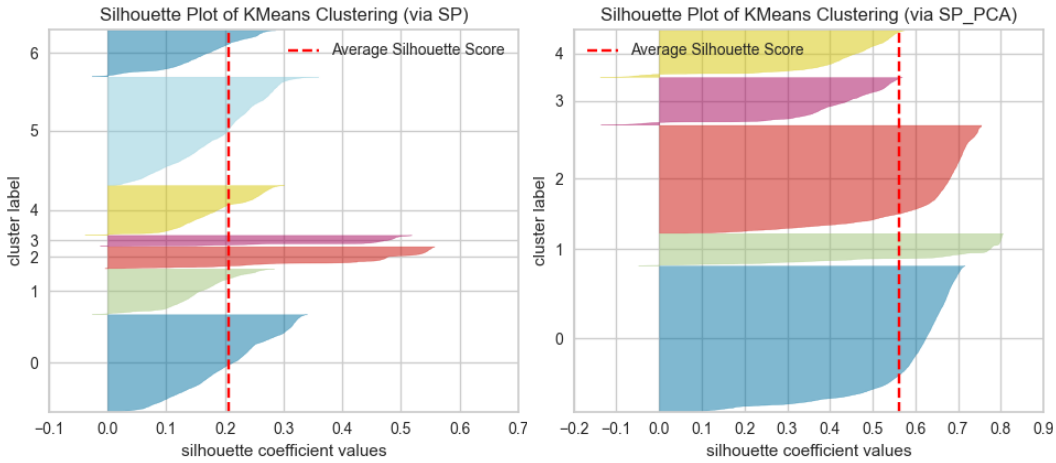

Figure 3.5: Silhouette analysis plot of KMeans clustering method utilizing the Standard Pipeline (SP), with and without the application of PCA.

Furthermore, the reduction in dimensionality based on PCA is applied before scaling and transformation as another version of this pipeline (SP_PCA). Figure 3.2 shows the total variability in the SP data is explained by the first 17 components. We examine the cumulative explained variance to select the most informative components, observing that the first three capture 99.4% of the total variance, where the PCA components are chosen accordingly.

The elbow analysis is depicted in Figure 3.3 for KMeans and Agglomerative clustering methods. It shows the distortion score and the training time, considering different values of cluster number k (2 to 18), whereas the dashed line represents the selected k for each pipeline. The performance of clustering methods utilizing the SP and SP_PCA data pipelines is compared in Table 3.1. The results show that using the PCA improves clustering quality for SP data pipeline, confirmed by all evaluation metrics for all clustering algorithms. This finding underscores the importance of incorporating PCA into the data analysis process to achieve optimal results.

The map in Figure 3.4 displays the intercluster distance in two-dimensional space for the KMeans and MeanShift clustering methods, utilizing SP data pipeline with and without PCA. The mapping is done by multidimensional scaling (MDS). This technique transforms high-dimensional data into a low-dimensional space while preserving the pairwise distances between data points as much as possible. It is crucial to note that just because two clusters overlap in the 2D space does not mean they also overlap in the original feature space. The clusters are sized according to membership, and

we can visually see the relative importance of each cluster accordingly. It can help determine the most significant clusters in the data and make decisions based on their characteristics.

Figure 3.5 shows silhouette plots for KMeans clustering for both data pipelines, where the dashed line represents the average scores. The average silhouette score for SP_PCA is higher than that for SP, which indicates that the clustering performance is better for the SP_PCA pipeline. It demonstrates the effectiveness of incorporating PCA components into the clustering process, as it helps reduce the dimensionality and capture more meaningful features.

Poly Pipeline (PP): The polynomial function generates new features in the poly pipeline by combining the numeric features of standard pipeline exclusion, transformation, and normalization methods based on a given polynomial degree. Suppose we have a data set with n observations and 2 original features, denoted as $X = [x_1, x_2]$. A new feature matrix $Xpoly$ can be generated by applying a polynomial function of degree $D = 2$ to X as follows:

$$X_{poly} = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2] \tag{10}$$

observe that a column of ones is included in $Xpoly$ to account for the intercept term in the model.

The polynomial function can help capture nonlinear relationships between the features, but including too many polynomial features can lead to overfitting and increases computational complexity. Therefore, it is crucial to carefully select the degree of the polynomial function (D) and

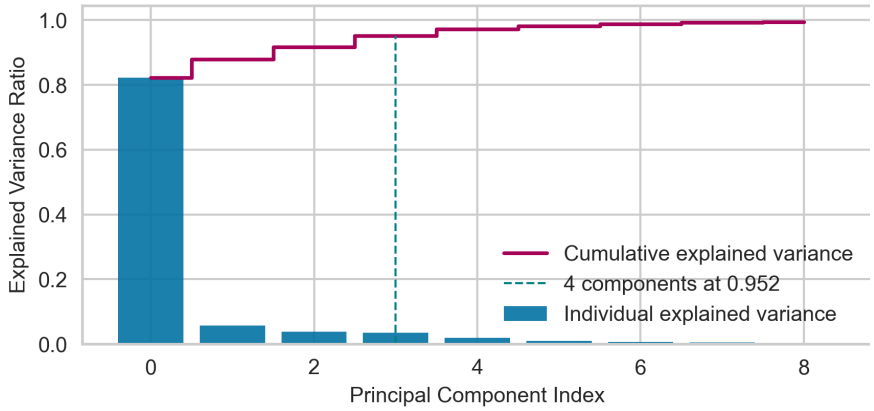

Figure 3.6: Explained variance plot for the transformed data using Poly Pipeline (PP), excluding the normalization and transformation methods

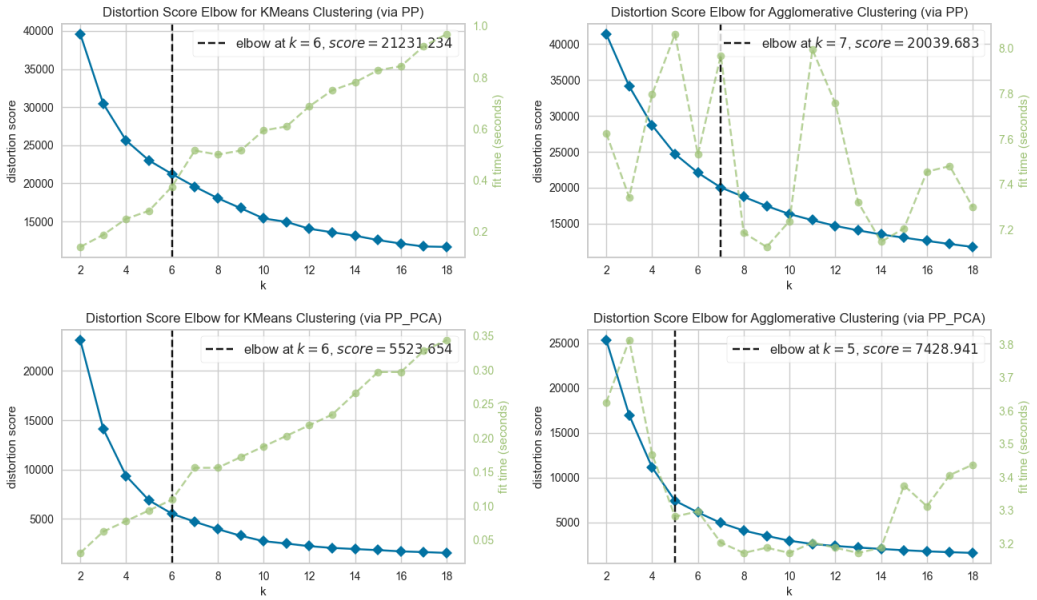

Figure 3.7: KElbow plots with distortion scores and training time for KMeans and Agglomerative clustering methods utilizing the Poly Pipeline (PP), with and without the application of PCA.

the features to be included in the model. For PP setups, the 22 features of SP (excluding the normalization and transformation methods) are used as input to a polynomial function with a degree of $D = 2$, producing 238 new features. The process of filtering new features involves applying specific variance and multicollinearity thresholds. In this case, the given thresholds are $VT = 0.05$ and $MT = 0.90$. Applying these thresholds aims to identify and keep only the most informative features. After applying the filters, the number of features is reduced to 113. These 113 features are the most informative ones identified through the filtering process.

Similar to SP, the dimensionality reduction is applied before the scaling and transformation as another version of the PP data pipeline (PP_PCA). Figure 3.6 shows the total variability in the PP data, observing that the first four capture 95.2% of the total variance. Thus, we only keep the first four components and disregard the others, which do not provide any extra informative variation.

Figure 3.7 illustrates the analysis of the elbow for KMeans and Agglomerative clustering methods, considering the PP data pipeline, with and without the use of PCA. This figure compares distortion scores and training times for various cluster numbers (k) ranging from 2 to 18. Notably, each pipeline's chosen cluster number is denoted by a dashed line. The performance of clustering methods utilizing the PP and PP_PCA data pipelines is compared in Table 3.2. The PCA based

Table 3.2: Comparison of Clustering Methods Utilizing the Poly Pipeline (PP), with and without the application of PCA.

Metrics	KMeans	Agglomerative	MeanShift	Pipeline
CHI	10537.35	9153.70	9997.13	PP_PCA
	2812.71	2582.08	637.91	PP
DBI	0.61	0.74	0.60	PP_PCA
	1.39	1.31	1.23	PP
SC	0.65	0.60	0.65	PP_PCA
	0.37	0.37	0.29	PP

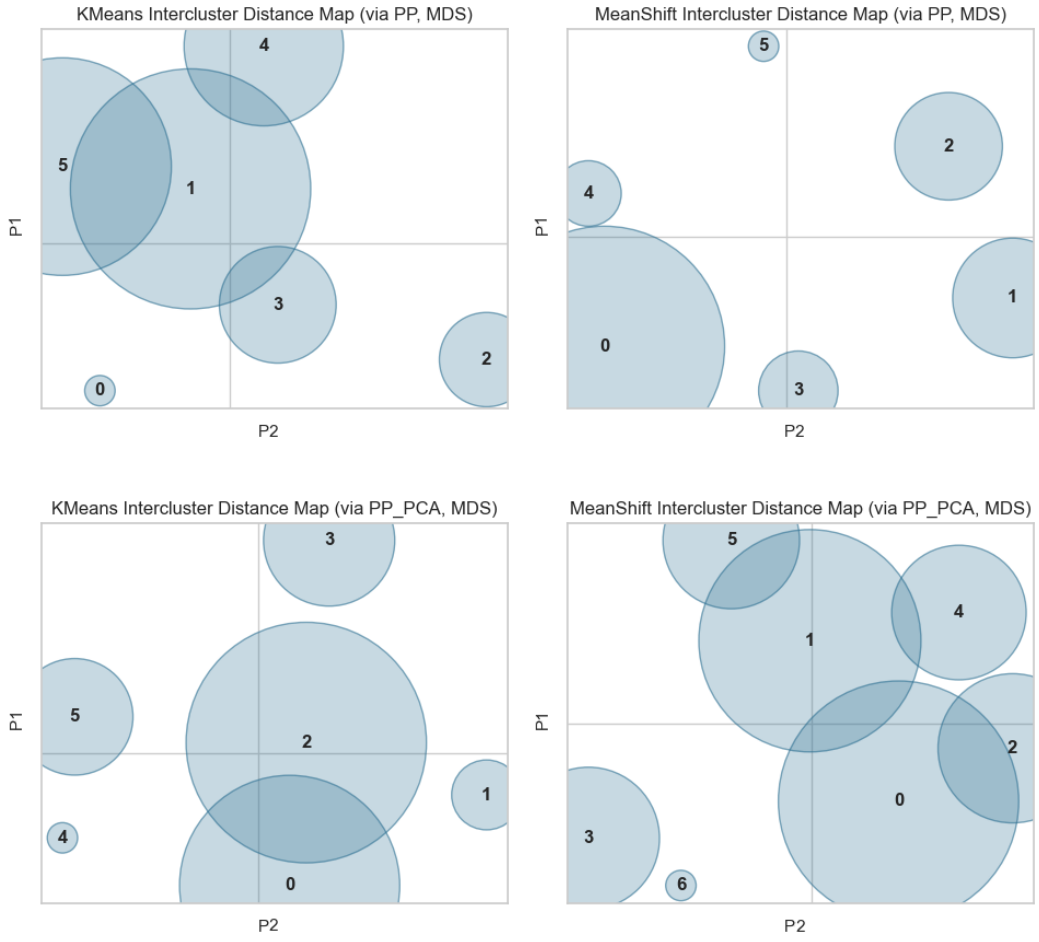

Figure 3.8: Intercluster distance map for KMeans and MeanShift clustering methods using the Poly Pipeline (PP), with and without the application of PCA embedded via the MDS into two features (P1, P2).

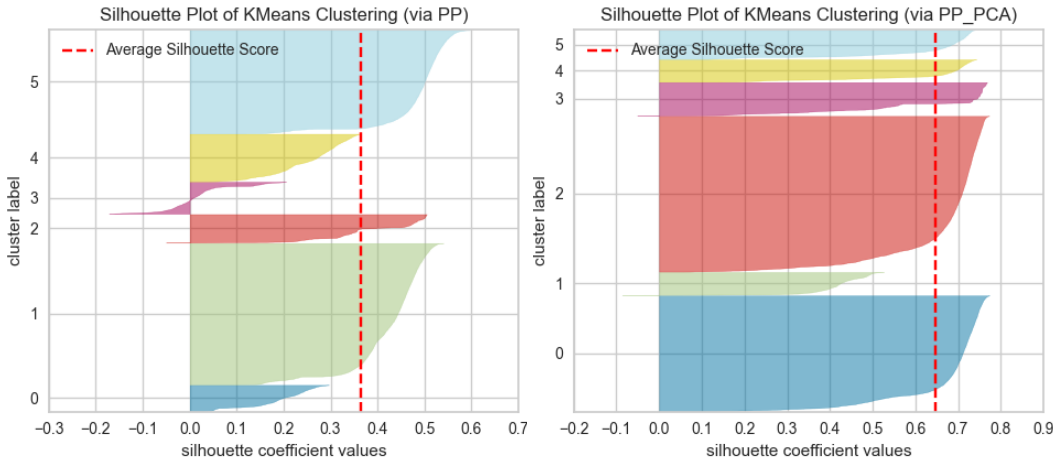

Figure 3.9: Silhouette analysis plot of KMeans clustering method utilizing the Poly Pipeline (PP), with and without the application of PCA.

PP data pipeline, like SP, considerably enhances the quality of clustering across all clustering algorithms. The evaluation metrics obtained in Table 3.2 provide sufficient evidence to support this assertion. Figure 3.8 shows the intercluster distance map in two-dimensional space for the KMeans and MeanShift clustering methods, and Figure 3.9 shows silhouette plots for KMeans clustering, using PP data pipeline with and without PCA.

In the comprehensive evaluation of clustering methodologies, the results accentuated the preeminence of the PP_PCA pipeline. When assessed based on performance metrics like the average silhouette score, PP_PCA consistently outperformed other pipelines such as SP, PP, and SP_PCA across all the clustering algorithms. The elevated efficacy of PP_PCA is attributed to its integration of PCA with polynomial functions. This unique amalgamation facilitates the capture of salient features and simultaneously reduces data dimensionality, enhancing the robustness and generalizability of the resulting clustering models. These findings emphasize the significance of careful preprocessing decisions and provide a path for improved VCI data analysis and system enhancements.

3.1.3 Conclusion

Workload categorization in cloud environments is imperative for consolidating workloads with homogenous characteristics. This study introduced a workload categorization method that underwent rigorous evaluation using real cloud workload. Our findings underscored the significance of

advanced data preprocessing and its seamless integration into clustering techniques, ensuring meticulous segmentation. Our exploration highlighted the superior performance achieved when dimensionality reduction converges with polynomial functions within the clustering pipeline. This study has been published in [100], and a similar study has been conducted and published in [101], investigating further clustering methods and data preparation configurations. Our next objective is to develop a clustering ensemble framework. The goal is to align the clustering results obtained from different pipelines used in base learners to discover powerful and precise clustering results. Combining these models creates a composite ensemble, which incorporates each member's advantages and ensures a more comprehensive and precise representation of workload categorization.

3.2 Ensemble Clustering for Multi-Perspective Workload Analysis

The accurate classification of cloud workloads is crucial for making informed decisions about load scheduling and resource allocation, leading to substantial improvements in the overall operational management of the infrastructure [14]. The sheer volume, velocity, and variety of workloads in large-scale cloud environments make it challenging for conventional clustering methods to produce accurate and meaningful results. Although these methods are somewhat effective, they have limitations when dealing with the diverse and multifaceted nature of cloud workloads [102]. The scalability and heterogeneity of the cloud environment frequently result in overlapping and nested clusters that are challenging to manage with conventional clustering methods.

In addition, hidden within the complexity of cloud workloads are multiple valid categorization perspectives that single clustering methods may overlook. This obfuscation compromises the efficiency of resource and task scheduling strategies in cloud data centers, resulting in suboptimal performance, increased costs, and diminished service quality. Therefore, a more adaptable and nuanced approach to workload categorization in cloud systems is necessary and timely.

Moreover, it is crucial to preprocess the observed data to ensure the precision of ML models. This process involves cleaning, integrating, transforming, and reducing the data to improve its quality and usefulness [8]. In cloud computing, acquiring knowledge can be challenging due to the diverse workloads being executed. In order to improve data quality and obtain valuable insights, it

is crucial to have adequate data preparation pipelines. Using efficient ones in clustering and prediction models can lead to more reliable and accurate results, optimizing the CRMS operational outcomes, which ensures efficient and effective resource utilization.

Given the limitations inherent in current clustering methods, this research aims to elevate workload segmentation by leveraging an advanced ensemble clustering approach. This study intends to refine the categorization process by employing enhanced data preprocessing and clustering methodologies tailored to cloud workloads' multifarious nature. The main contributions of this study can be summarized as follows.

- The formulation of a novel ensemble clustering technique that amalgamates diverse data preprocessing pipelines with varied base clustering learners, thereby ensuring a comprehensive understanding of intricate cloud workload patterns.
- Empirical validation of the proposed technique using real-world trace data derived from the Microsoft Azure workload, attesting to its robustness, precision, and relevance in identifying complex workload attributes.
- Introduction of a unique scoring method that integrates the Silhouette Coefficient (SC), Calinski-Harabasz Index (CHI), and Davies-Bouldin Index (DBI) metrics. This score facilitates the discerning selection of optimal clustering models and preprocessing mechanisms, illuminating the nuanced interplay between distinct pipeline configurations and arrangements.
- Detailed analysis of the interactions between different clustering algorithms and a combination of transformation and normalization methods, offering profound insights into their collective influence on the workload categorization process efficacy.
- To increase the adaptability of the reduction of data dimension, we combine the Kneedle method [91] with PCA, as detailed in Algorithm 3.5. This novel fusion approach autonomously determines the optimal PCA components that can be used count the inflection in the cumulative variance curve, obviating predefined thresholds and enhancing processing effectiveness.

The significance of this study lies in its potential to markedly enhance workload segmentation efficiency, paving the way for nuanced analyses and optimized cloud infrastructure management.

By highlighting the efficacy of ensemble clustering in addressing the multifaceted nature of cloud workloads, this research seeks to propel advancements in the field. Furthermore, the proposed ensemble clustering approach underscores a promising avenue for superior cloud workload clustering. The proposed approach exemplifies the integration potential of sophisticated clustering strategies within real-world cloud resource management contexts.

3.2.1 Proposed Ensemble Clustering Approach

Data preprocessing pipelines are a systematic sequence of steps used to clean and transform raw data into a format suitable for analysis, ensuring consistent and efficient data processing [8]. On the other hand, ensemble clustering merges multiple clustering algorithms or configurations to categorize data [103]. This combination aims to maximize the strengths and minimize the weaknesses of individual methods, yielding more accurate and robust groupings of data points. Both concepts underscore the value of integrating various methods to achieve enhanced results. The proposed methodology involves the integration of base clustering pipelines, which comprise multiple algorithms and data preprocessing pipelines, to enhance the categorization of cloud workloads. A graphical illustration of the ensemble clustering approach is provided in Figure 3.10, while Table 3.3 presents the notation definitions.

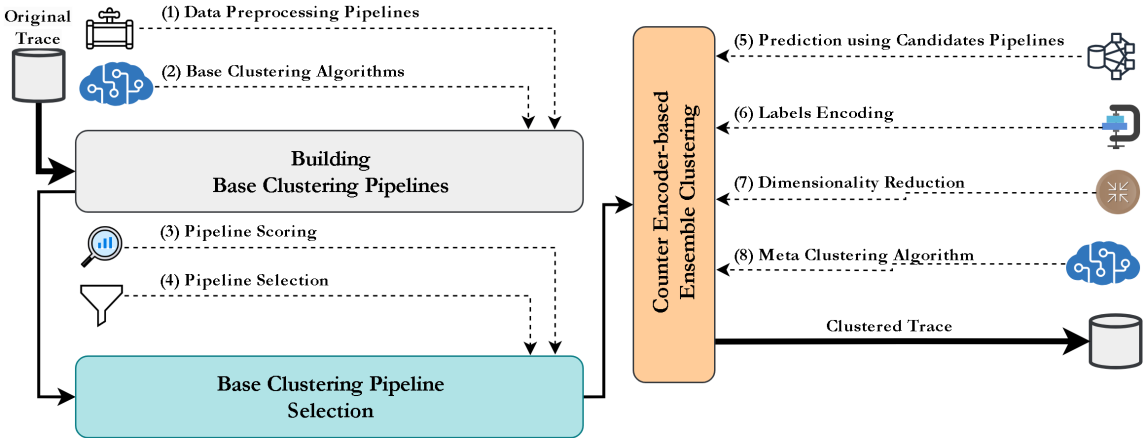

Figure 3.10: Visual representation of the proposed ensemble clustering model showcasing the processes the original workload trace goes through to perform the segmentation process.

Table 3.3: Table of Definitions

Symbol	Definition
X	Workload trace data matrix
P	Set of data preprocessing processes
B	Set of base clustering algorithms
CP	List of combined clustering pipelines
CP_{ij}	Clustering pipeline combining the i-th of P with the j-th of B
X'_{ij}	Data post application of CP_{ij}
L_{ij}	Clustering labels from CP_{ij}
$SC_v, CHI_v,$ and DBI_v	SC, CHI, DBI score lists for CP
$SC_{ij}, CHI_{ij},$ and DBI_{ij}	Evaluation scores for CP_{ij}
$SC'_{ij}, CHI'_{ij},$ and DBI'_{ij}	Normalized scores of CP_{ij}
CS	Combined score list for CP
CS_{ij}	Combined score for CP_{ij}
SP	Pipeline selection criteria
T	Threshold based on SP
CP'	Selected top-performing pipelines
M	Meta clustering algorithm
k	Specified cluster count
k'	'Knee' point in variance data
a	Flag for applying PCA to base clustering outputs
X''	Input data matrix for PCA
u	Flag for using Kneedle algorithm [91] in PCA
$\mathbb{E}$	PCA variance ratio
v	Explained variance threshold for PCA
$\mathbb{C}$	Cumulative variance ratio in PCA
L	Label list from selected top-performing pipelines CP'
L^T	Transposed version of L
L_{en}	Counter-encoded labels of L^T
L'_{en}	PCA-transformed L_{en}
L'	Final ensemble labels

The approach begins by constructing base clustering pipelines, each comprising a unique combination of data preprocessing and base clustering algorithms. These candidate pipelines are then evaluated and selected based on their partitioning quality. The selected pipelines generate clusters whose labels are encoded using a count encoder. A PCA-based dimensionality reduction can be applied to handle potential high dimensionality in the label space. Lastly, a meta-clustering algorithm consolidates the results from the selected base clustering pipelines, producing a final ensemble clustering output. Thus, this approach leverages the strengths of various clustering algorithms and data preprocessing methods, providing robust and reliable categorization of cloud workloads.

3.2.1.1 Building Base Clustering Pipelines

The Algorithm 3.3 outlines the first significant step in our ensemble clustering approach - *Building Base Clustering Pipelines*. This step is critical to preparing a comprehensive set of base clustering pipelines that constitute the building blocks of our ensemble approach. These pipelines combine each data preprocessing method, denoted by P, with each base clustering algorithm, denoted by B. Both P and B are lists; P consists of different data preprocessing pipelines such as normalization, standardization, PCA-based dimensionality reduction, and other feature engineering methods as required based on the data structure. While B comprises various base clustering algorithms such as the KMeans, Agglomerative, and MeanShift algorithms as required based on the given data.

In detail, the algorithm begins by initializing an empty list CP to hold the resulting base clustering pipelines (Line 1). It then proceeds with two nested loops to iterate over every possible combination of elements from lists P and B (Lines 2 to 7). In each iteration, a base clustering algorithm B_j is combined with a preprocessing data pipeline P_i to form a new clustering pipeline, denoted by CP_{ij} (Line 4). The clustering pipeline is basically a sequence of preprocessing steps P_i that need to be applied to the data X and ends with the clustering algorithm B_j. Each created pipeline CP_{ij} is then added to the list CP (Line 5), which includes a pool of pipeline combinations.

This systematic process ensures that every potential combination of preprocessing and base clustering methods is considered, thus increasing the likelihood of finding a set of pipelines that offer high-quality clustering results. At the end of the algorithm, the list CP is returned (Line 8).

The list CP contains base clustering pipelines to be scored and evaluated in the next step of the ensemble clustering approach, known as *Base Clustering Pipeline Selection*. This step is not a mere cursory evaluation. It is a pivotal juncture where the theoretical constructs of our approach meet empirical validation. We undertake a rigorous quality assurance process by scoring and evaluating the pipelines stored in CP. This step ensures that only the most efficacious pipelines, which are in harmony with the intrinsic data structure, are retained. Such precision-driven selection amplifies the robustness of our ensemble clustering, ensuring that our final model is not just a conglomeration of various clustering pipelines, but a synergized ensemble informed by data-driven insights. Using this selection mechanism, we can confidently guarantee the highest accuracy and reliability in the

Algorithm 3.3 Building Base Clustering Pipelines

Input
X Original workload trace data matrix
P $= P_1, P_2, ..., P_s$ Data Pipeline List
B $= B_1, B_2, ..., B_c$ Base Clustering Algorithms

Output
$CP = CP_{ij} : CP_{ij}$ is a clustering pipeline that combines P_i and B_j ▷ List of clustering pipelines

$CP \leftarrow$ Empty list ▷ Initialize the pipeline list
for each P_i in P **do** ▷ Iterate over preprocessing methods
  for each B_j in B **do** ▷ Iterate over base algorithms
    Combine B_j with P_i based on X and obtain CP_{ij} ▷ Create a clustering pipeline CP_{ij}
    Add CP_{ij} to CP ▷ Add the pipeline to the list
  end for
end for
return CP ▷ Return the list of pipelines

Algorithm 3.4 Base Clustering Pipeline Selection

Input
X Original workload trace data matrix
P $= \{P_1, P_2, ..., P_s\}$ Data Pipeline List
B $= \{B_1, B_2, ..., B_c\}$ Base Clustering Algorithms
SP Selection policy

Output
CP'List of selected clustering pipelines

$CP \leftarrow$ Call Algorithm 3.3 with X, P, and B ▷ Construct base clustering pipelines
Initialize SC_v, CHI_v, DBI_v as empty lists ▷ Initialization of lists to store clustering validity scores
for each CP_{ij} in CP **do** ▷ Iterate over all pipeline combinations
  Fit CP_{ij} on X to get labels L_{ij} and transformed data X' ▷ Process data using current pipeline
  Calculate SC_{ij}, CHI_{ij}, and DBI_{ij} scores using X' and L_{ij} ▷ Evaluate the base clustering pipelines
  Append SC_v, CHI_v, and DBI_v with SC_{ij}, CHI_{ij}, and DBI_{ij}, respectively.
end for
Initialize CS as empty list ▷ Prepare for combined validity scores
for $(SC_{ij}, CHI_{ij}, DBI_{ij})$ in zip(SC_v, CHI_v, DBI_v) **do** ▷ Iterate over clustering validity scores
  Normalize the validity scores according to Equations 11, 12, and 13 to obtain SC'_{ij}, CHI'_{ij}, and DBI'_{ij}.
  Compute Combined Score CS_{ij} according to Equation 14 ▷ Aggregate the normalized scores
  Append CS_{ij} to CS ▷ Add aggregated score to the list
end for
Calculate threshold T using the selection policy SP applied on CS ▷ Determine cut-off score for selection
$CP' \leftarrow$ base clustering pipelines with CS above T from CP ▷ Filter pipelines surpassing the threshold
return CP' ▷ Return the subset of top-performing pipelines

clustering process, placing our approach above conventional ensemble clustering methods.

3.2.1.2 Base Clustering Pipeline Selection

Algorithm 3.4 presents the second major step of our proposed ensemble clustering approach - *Base Clustering Pipeline Selection*. The primary aim of this step is to identify the top-performing base clustering pipelines that will contribute to the final ensemble clustering solution. This process is achieved by evaluating each pipeline's clustering performance on the trace data, X, using a Combined Score (CS) of a set of evaluation metrics. The clustering pipelines are then filtered based on a threshold T calculated based on a criterion defined by the selection policy, SP, such as the top

Algorithm 3.5 PCA-based Dimensionality Reduction

Input
X'' $= \{X''_1, X''_2, ..., X''_n\}$ Input Data Matrix
u Use Kneedle True or False
v Explained Variance Threshold

Output
X' Transformed X'' after applying PCA

Apply PCA on X'' and get explained variance ratio $\mathbb{E}$ ▷ Apply PCA and Compute variance ratios for data
Cumulative Explained Variance Ratio $\mathbb{C} \leftarrow$ Cumulative Sum of $\mathbb{E}$ ▷ Get cumulative sum of variances
if u **then** ▷ Check if Kneedle method is to be used
    Use Kneedle to find knee k' from $\mathbb{C}$ ▷ Identify the 'knee' point in the cumulative variance
    if knee k' is None **then** ▷ Check if no knee point was found
        $k \leftarrow$ first index where $\mathbb{C}$ exceeds v ▷ Determine number of components based on threshold
    end if
else
    $k \leftarrow$ first index where $\mathbb{C}$ exceeds v ▷ Determine number of components directly based on threshold
end if
Apply PCA with k' components to X'' and get X' ▷ Apply dimensionality reduction
return X' ▷ Return transformed data

Algorithm 3.6 Counter Encoder-based Ensemble Clustering

Input
X Original workload trace data matrix
CP' List of selected base clustering pipelines
M Meta Clustering Algorithm
k number of clusters
a Apply PCA to base Clustering Algorithms Results {True or False}
u Use Kneedle {True or False}
v Explained Variance Threshold

Output
L' Ensemble Clustering Labels

if k is not None and has $M.n\_clusters$ **then** ▷ Check if a specific number of clusters is provided
    $M.n\_clusters \leftarrow k$
end if
L as empty list ▷ Initialize list for base clustering labels
for each CP'_{ij} in CP' **do** ▷ Iterate over selected base clustering pipelines
    Fit CP'_{ij} on X and get labels L_{ij} ▷ Compute labels using each pipeline
    Append L_{ij} to L
end for
$L^T \leftarrow$ transpose of L ▷ Transpose to structure data for encoding
Encode L^T using **counter encoding method** and get encoded labels L_{en} ▷ Counter encode the labels
if a **then** ▷ Check if PCA is to be applied
    $L'_{en} \leftarrow$ Call Algorithm 3.5 with L_{en}, u, and v
else
    $L'_{en} \leftarrow L_{en}$
end if
Fit M on L'_{en} ▷ Apply meta clustering on encoded labels
$L' \leftarrow M$.labels ▷ Retrieve the final ensemble labels
return L' ▷ Return the final ensemble clustering labels

models m or the medium score threshold.

The choice of clustering evaluation metrics is grounded in thoughtful deliberation. We have chosen SC, CHI, and DBI metrics for their proven track record and effectiveness across several studies, detailed in Section 3.1.2.1. These metrics provide a comprehensive assessment of clustering quality: SC measures the tightness and isolation of clusters, CHI evaluates intra-cluster cohesion,

and DBI inspects the distinctness between clusters. By adopting a multifaceted approach to the evaluation process, we can ensure that clustering results are robust and holistic.

To initiate, the algorithm first constructs the base clustering pipelines by invoking Algorithm 3.3, with the original data X, the list of data preprocessing pipelines P, and the base clustering algorithms B as input parameters (Line 1). Following this, it initializes three empty lists, SC_v, CHI_v, and DBI_v, which will, respectively, store the SC, CHI, and DBI scores for each base clustering pipeline (Line 2). In detail, the algorithm performs the following operations for each pipeline CP_{ij} in the constructed list of base clustering pipelines, CP. It first fits the pipeline CP_{ij} on the original data X to produce the cluster labels L_{ij} and transformed data X' (Line 4). It then calculates the SC, CHI, and DBI scores using the transformed data X' and labels L_{ij}, and appends these scores to the respective lists SC_v, CHI_v, and DBI_v (Lines 5 to 6).

Once the evaluation metrics have been calculated for each pipeline, the algorithm calculates a CS (Lines 8 to 13). It first normalizes the SC, CHI, and DBI values using Equations 11, 12, 13, to generate normalized scores $SC'ij$, $CHI'ij$, and $DBI'ij$ respectively. Then, it calculates the combined score $CSij$ for each pipeline according to Equation 14. Following the computation of the combined scores, the selection policy SP is applied to the list CS to determine a threshold T to select the best performing pipelines (line 14). Those pipelines whose combined score CS is above the threshold T are selected and stored in the list CP' (Line 15).

It is crucial to specify the precise context in which the proposed combined score operates optimally. This metric has been judiciously designed to amalgamate the outcomes, namely SC_v, CHI_v, and DBI_v, derived from various candidate clustering pipelines. Through rigorous normalization procedures, the inherent diversity in the value ranges of these metrics is harmonized to fit a consistent scale, ensuring a holistic and equitable synthesis into the combined score. It is noteworthy, however, that this combined metric is tailored for appraising an ensemble of clustering pipelines. For evaluating an individual clustering algorithm, such as the final ensemble clustering algorithm, the utility of the combined score is constrained. In such instances, the conventional metrics, specifically SC, CHI, and DBI, proffer a more immediate and illustrative assessment.

In summary, Algorithm 3.4 effectively ranks and selects the base clustering pipelines based on

their clustering performance, represented by a combined score of multiple validity indices. The output is a subset of the original list of base clustering pipelines, CP', that contains the top-performing pipelines selected for the final ensemble clustering.

3.2.1.3 Counter Encoder-based Ensemble Clustering

Firstly, in Algorithm 3.6 - *Counter Encoder-based Ensemble Clustering*, it takes as input the original workload trace data matrix, X, the list of selected base clustering pipelines, CP', the meta clustering algorithm, M, and a few additional parameters. If a specific number of clusters, k, has been provided and the meta clustering algorithm M has an attribute $n\_clusters$, the algorithm sets $M.n\_clusters$ to k (Lines 1 to 3). For each selected base clustering pipeline, $CP'ij$, in the list CP', the algorithm fits the pipeline on the data X and retrieves the cluster labels Lij (Line 6). These labels are added to an initially empty list, L (Line 7). After obtaining the cluster labels from all selected pipelines, the algorithm transposes the list L to L^T (Line 9). The counter-encoding method is then applied to L^T, resulting in the encoded labels L_{en} (Line 10).

Following the counter encoding process, Algorithm 3.5 - *PCA-based Dimensionality Reduction* is applied if the flag a is set to True. As an optional process, this algorithm leverages PCA to reduce the dimensionality of the encoded labels, L_{en}, and transform them into a lower-dimensional space that maintains most of the original data variance. There are two recommended methods to determine the optimal PCA components. The first involves utilizing a Kneedle algorithm [91] to identify the inflection point. Alternatively, a threshold for the explained variance, denoted by the v, can be set to attain the desired outcome. Both approaches are practical and depend on the specific needs of the analysis. However, Algorithm 3.5 applies PCA on L_{en}, yielding an explained variance ratio, $\mathbb{E}$ (Line 1). The cumulative explained variance ratio, $\mathbb{C}$, is calculated as the cumulative sum of $\mathbb{E}$ (Line 2). If the use Kneedle flag, u, is set to True, a Kneedle algorithm finds a knee point, k', from $\mathbb{C}$ (Line 4). If no knee point is found, or if u is set to False, k' is determined as the first index where $\mathbb{C}$ exceeds the explained variance threshold, v. Lastly, PCA is applied with k' components to L_{en}, resulting in the transformed encoded labels L'_{en} (Line 11).

In Algorithm 3.5, the number of PCA components is determined primarily using the Kneedle algorithm. This technique is tailored to detect the knee point in the cumulative variance curve,

configured as ”concave” and ”increasing”. The knee represents a critical inflection where further inclusion of components offers diminishing explanatory returns. However, we adopt an intuitive selection approach when the Knee Locator fails to identify a clear knee. We can select a variance threshold v, typically between 95% and 99%, which a systematic trial-and-error tuning process can guide to achieve optimal performance.

After the dimensionality reduction, in Algorithm 3.6, the final step is to fit the meta clustering algorithm M on L'_{en} (Line 16). The ensemble clustering labels, L', are obtained as M labels (Line 17). The algorithm then returns these labels (Line 18), thus completing the ensemble clustering process based on multi-perspective data preprocessing and clustering pipelines.

In summary, the *Counter Encoder-based Ensemble Clustering* algorithm and the *PCA-based Dimensionality Reduction algorithm* work synergistically to transform the base clustering results into a suitable format, reduce their dimensionality and generate the final ensemble clustering. This combination provides a practical and adaptable approach to ensemble clustering based on multi-perspective pipelines. Thus, employing various clustering algorithms in this context can significantly improve overall performance compared to relying solely on a singular clustering algorithm.

3.2.1.4 Mathematical Formulations

Our ensemble clustering method uses a set of mathematical procedures to achieve robust clustering outcomes. These processes encompass pipeline evaluation, score normalization, counter encoding, and the final meta-clustering. In order to measure the effectiveness of the essential clustering pipelines, we utilize three commonly used clustering metrics (SC, CHI, and DBI). These metrics are detailed in Section 3.1.2.1. Each of these metrics provides valuable insights into the quality of the clustering generated by a base clustering pipeline CP_{ij} on a given data set X, resulting in a corresponding label set L_{ij}. These metrics for each pipeline coalesce into metric vectors, namely SC_v, CHI_v, and DBI_v. However, these metrics vary in their ranges, making direct comparisons challenging. Thus, we normalize these metrics to ensure they reside on a uniform scale, as follows:

$$SC'ij = \frac{SCij - \min(SC_v)}{\max(SC_v) - \min(SC_v)} \tag{11}$$

$$CHI'ij = \frac{CHIij - \min(CHI_v)}{\max(CHI_v) - \min(CHI_v)} \tag{12}$$

$$DBI'ij = 1 - \frac{DBIij - \min(DBI_v)}{\max(DBI_v) - \min(DBI_v)} \tag{13}$$

Each normalized score now falls within the interval [0, 1], facilitating their amalgamation into a combined score CS_{ij} using weights α, β, and γ for each respective metric. The optimal value corresponds to the maximum score in the context of these normalized scores and the resulting combined one. The maximum value represents the highest performance level achievable among the clustering pipelines. It serves as the ultimate goal in the pipeline selection process. The combined score is calculated as follows:

$$CS_{ij} = \alpha \cdot SC'ij + \beta \cdot CHI'ij + \gamma \cdot DBI'_{ij} \tag{14}$$

where the weights α, β, and γ reflect the relative importance of each metric in the combined score. With CS_{ij} for each pipeline, we choose base clustering pipelines that surpass a certain threshold T for the final ensemble clustering step. The user-defined selection policy SP influences the threshold T by using, for instance, the median of the CS vector or top n clustering pipelines.

Following the selection of the base clustering pipelines, their produced labels are encoded using the *Counter Encoding* method. This method treats each unique set of labels across the pipelines as a separate group. It replaces them with the count of their occurrences, effectively transforming the categorical labels into numerical features. These encoded labels are then subjected to PCA for dimensionality reduction. PCA is a sophisticated mathematical technique that can minimize the number of correlated variables in a given data set. This process transforms these variables into a smaller set of uncorrelated variables called principal components. The first principal component captures as much data variability as possible, and each subsequent component captures as much of the remaining variability as possible. It is an effective tool for simplifying complex data sets and extracting meaningful information.

Finally, these transformed and reduced labels serve as input to the chosen meta-clustering algorithm M, which applies its mathematical formulation to produce the final ensemble clustering labels L'. The specific formulation varies based on the type of meta-clustering algorithm used. Overall, the mathematical formulations in the proposed ensemble clustering approach are span normalization, weighting, encoding, dimensionality reduction, and clustering, creating a robust and adaptive framework for clustering tasks.

3.2.1.5 Time and Space Complexity Analysis

Understanding the time and space complexity of the Counter Encoder-based Ensemble Clustering method is crucial for evaluating its efficiency and scalability. The algorithm consists of multiple steps, each with its associated computational complexity. The time complexity of a base clustering pipeline (CP_{ij}) is the sum of the time complexities of its data preprocessing steps (P_i) and its base clustering algorithm (B_j). Similarly, the space complexity of a base clustering pipeline is the maximum of the space complexities of its data preprocessing steps and its base clustering algorithm.

The counter-encoding step transforms the categorical labels into numerical values. The time complexity of this operation is $O(n \cdot m)$, where n is the number of data instances and m is the number of selected base clustering pipelines. The space complexity remains $O(n \cdot m)$ as well, as we need to store the encoded labels for each data instance from each pipeline. If PCA is applied to reduce the dimensionality of the encoded labels, the time complexity would be $O(n \cdot m^2)$, as PCA requires calculating the covariance matrix and performing eigenvalue decomposition. The space complexity here depends on the number of principal components selected but would be at most $O(n \cdot m)$. Upon transforming the labels, the time complexity of fitting the meta-clustering algorithm M is subject to variation based on the selected algorithm, where the space complexity is $O(n)$ as we store the final clustering labels for each data instance.

By denoting the time complexity of M as T_M, we can represent a formal representation as follows. For instance, if M were to be instantiated as the KMeans algorithm, T_M would manifest as $O(I \times k \times n)$, where I represents the number of iterations, k the number of clusters, and n the number of data instances. Conversely, if M was a hierarchical clustering algorithm, the upper bound on the complexity could be $O(n^3)$, albeit more efficient variations could proffer a complexity

of $O(n^2 \log n)$. Thus, the definitive characterization of T_M remains contingent upon the specific algorithmic nature of M. In practical applications, it is crucial to recognize the variability and choose an appropriate value for M that balances computational efficiency and clustering accuracy while also considering the operational constraints of the application domain.

The overall time complexity of the ensemble clustering approach is thus the time complexity of applying all base clustering pipelines plus the time complexity of the meta-clustering algorithm. The overall space complexity is the maximum of the space complexities of all base clustering pipelines and the space complexity of the meta clustering algorithm. Nevertheless, it is essential to note that the specific time and space complexity can vary greatly depending on the specific details of the data sets and the algorithms used. While the method may have relatively high computational requirements due to multiple stages of computations, its advantages in terms of robustness and flexibility can justify the computational cost in many practical scenarios, mainly when efficient clustering algorithms are used for base clustering pipelines and the meta-clustering process.

3.2.2 Experimental Setup and Evaluation

Our proposed ensemble clustering approach for workload categorization is evaluated using actual cloud data center workloads. Different clustering algorithms and data pipeline setups are employed. Additionally, various intrinsic performance metrics are used to assess the quality of the clustering results. More details are outlined as follows.

3.2.2.1 Workload Description and Initial Data Preparation

The data set used in this study is derived from the Azure Public Data set v2 [96]. Given the extensive size of the data set, we employed a stratified sampling to ensure manageable computational requirements while maintaining representativeness. A sample of 15,000 VMs is chosen for our examination, comprising an even distribution across the three VM types: Interactive, Delay-insensitive, and Unknown. Specifically, 5,000 instances of each type are randomly selected, ensuring a diverse and comprehensive representation of workloads for our clustering tasks.

Standard Preprocessing Pipeline: A standardized preprocessing pipeline is employed to prepare the data set for subsequent analysis. It identifies additional characteristics, including the difference between the peak and average CPU usage (referred to as "Diff 1") and the difference between the 95^{th} percentile of maximal and average CPU usage (referred to as "Diff 2"). The pipeline also extracts various representative time-related data from the VM creation and deletion timestamps, including the year, month, day, hour, minute, and weekday indexes.

The categorical variables are then converted to numerical values using three types of encoders.

- *Ordinal Encoder:* This encoder assigns a distinct integer to each CPU core and memory bucket category based on their respective sizes. This encoder is commonly used in data analysis to simplify ordinal categories while preserving their magnitudes.

- *One-hot Encoder:* This encoder generates a binary representation for each VM type. This approach ensures that nominal categories are structured and organized, facilitating efficient handling and processing.

- *Counter Encoder:* This encoder replaces each category label with a numerical count of its appearances while maintaining the frequency-based ordering of the categories. Using this approach, we can better understand the distribution of these high cardinal nominal indexes within the data set, facilitating more representative numerical details.

Finally, we identified features with high inter-correlation using the absolute Pearson correlation coefficient. Any features with a correlation value greater than the 0.99 threshold are deemed excessively correlated; of these, only the first feature in each correlated set is retained. Additionally, we eliminated features with zero variance, as they fail to provide meaningful differentiation. These feature selection measures ensure the retention of only the most relevant and informative features, thereby enhancing the efficiency and effectiveness of subsequent analyses.

3.2.2.2 Experimental Setup

This section details the methodologies employed for constructing and assessing clustering pipelines. Initially, the base clustering pipelines and selection setups are created using a combination of transformation and normalization methods and a PCA-based dimensionality reduction technique. Three

Table 3.4: The number of PCA components and cumulative explained variance utilizing various combinations of transformation and normalization methods.

P	Transformation	Normalization	PCA Components	$\mathbb{C}$
P_1	None	None	3	0.99
P_2	None	Z-Score	8	0.84
P_3	None	Min-Max	8	0.89
P_4	None	MaxAbs	8	0.89
P_5	None	Robust	6	0.86
P_6	Yeo-Johnson	None	4	1.00
P_7	Yeo-Johnson	Z-Score	8	0.89
P_8	Yeo-Johnson	Min-Max	8	0.91
P_9	Yeo-Johnson	MaxAbs	8	0.91
P_{10}	Yeo-Johnson	Robust	8	0.90
P_{11}	Quantile	None	7	0.92
P_{12}	Quantile	Z-Score	8	0.87
P_{13}	Quantile	Min-Max	7	0.92
P_{14}	Quantile	MaxAbs	7	0.92
P_{15}	Quantile	Robust	6	0.90

clustering algorithms serve as base clustering methods, and the optimal pipelines are chosen using a combined score derived from various normalized metric scores. The selected pipelines form a refined list, serving as inputs to the ensemble clustering task. Furthermore, we introduce the ensemble clustering approach setups based on the counter encoder. Finally, we introduce a strategy to select the ideal number of clusters using a distortion score, quantifying the divergence between data points and their respective cluster centers. The optimal elbow value, indicative of the optimal cluster count, is automatically identified through the knee point detection algorithm, providing a complete setup for our ensemble clustering task.

Base Clustering Pipelines and Selection Setups: The clustering pipelines are built on different data pipeline setups, each utilizing a standard preprocessing pipeline. The pipelines are formulated with PyCaret, a top-tier Python ML library [104]. These pipelines use various transformation and normalization techniques and instances where they are not used.

Among the transformations utilized is the Yeo-Johnson method, a power transformation designed to stabilize variance and make the data more closely follow a Gaussian distribution. This transformation is remarkably versatile as it can handle zero and negative values, making it suitable for different data aspects in many data sets. Another transformation employed is the Quantile

Transformation, which transforms the features to follow a uniform or a Gaussian distribution. This method is beneficial in mitigating the effects of outliers, hence improving the performance of subsequent clustering algorithms [9].

For normalization, we employed multiple techniques to ensure data consistency and comparability. The Z-Score normalization standardizes the features by removing the mean and scaling to the standard deviation, which results in features with a mean of zero and a variance of one. Min-Max normalization scales the data between a specific range, in our use case [0 to 1], ensuring every feature equivalently contributes to distance computations in clustering. MaxAbs normalization scales each feature by its maximum absolute value, ensuring that the maximal absolute value of each feature in the training set will be 1.0. It is noteworthy that MaxAbs does not shift or center the data, preserving any inherent sparsity in the data set. The Robust normalization method, resilient to outliers, scales features using the median and the interquartile range, making it an optimal choice for data sets with notable outliers [9].

These setups may differ in the inclusion/exclusion of specific methods for each base clustering pipeline. The data pipelines incorporate Algorithm 3.5 to reduce the dimensionality of the original data. It applies a Kneedle method by setting True to the u parameter to determine the PCA components. In situations where the identification of the knee index is not feasible, we set a threshold of 95% explained variance v to attain the intended objective. This crucial step entails transforming the data into a lower-dimensional space while preserving most of the original variance, considering the selected transformation and normalization setups.

Table 3.4 shows the results of PCA components and cumulative explained variance ($\mathbb{C}$) for various combinations of transformation (None, Yeo-Johnson, and Quantile) and normalization methods (None, Z-Score, Min-Max, MaxAbs, and Robust). This table provides insights into how different combinations of methods can affect the optimal number of PCA components and the $\mathbb{C}$ that indicates the variance in the data explained by the selected components. It allows for comparing the performance of different setups and identifying the configurations that yield higher explained variance, serving as a reference for understanding the influence of the given setups.

Three clustering algorithms (KMeans [97], Hierarchical clustering - Agglomerative (with an early stop of tree construction) [98], and MeanShift [99]) are employed as base clustering methods

using the implementation of sklearn [9]. Accordingly, Algorithm 3.3 generates all possible clustering pipelines based on the given setups. This process is considered the first step in the Algorithm 3.4 to formulate a comprehensive set of pipelines incorporating various setups.

In our selection of clustering algorithms, we are driven by a desire to encompass diverse algorithmic philosophies that cater to varied data characteristics and use cases. The KMeans algorithm, as elucidated by Sculley [97], offers a centroid-based clustering mechanism renowned for its scalability and efficiency, making it indispensable for large data sets. The Hierarchical clustering - Agglomerative method, showcased in [98], lends a tree-structured representation qualified for interpreting hierarchical relationships within data. Lastly, MeanShift, referenced in [99], complements the prior methods by offering a density-based clustering paradigm, adept at discerning clusters of arbitrary shapes, and excelling in scenarios with intricate data densities. These methods provide a robust and comprehensive toolkit, each contributing a unique perspective and addressing specific clustering challenges.

Nevertheless, Algorithm 3.4 selects the optimal base clustering pipelines using a combined score. Equation 14 calculates this score, with weightings assigned to each normalized metric score of the base clustering pipelines. The weights α, β, and γ are set to 34%, 33%, and 33%, respectively, indicating equal importance for all contributed metrics. In order to be considered for selection, pipelines must exceed a predetermined threshold T, which is determined by the selection policy SP. This study has set T to the median of the combined score vector, allowing for a flexible practice in selecting the most appropriate pipelines for the final ensemble clustering task, enabling consideration of various data preprocessing perspectives.

In summary, Algorithm 3.3 generates a list of all potential clustering pipelines according to the given specifications, denoted as CP. Algorithm 3.4 accurately evaluates and chooses the best base clustering pipelines by considering their performance in various validity indices. The outcome is a refined list, denoted as CP', which comprises the top-performing pipelines chosen for the ultimate ensemble clustering solution.

Counter Encoder-based Ensemble Clustering Setup: As a meta-clustering method, we examined the same clustering algorithms used in building the base clustering pipelines (KMeans,

Agglomerative (with an early stop of tree construction), and MeanShift) based on the implementation of sklearn [9]. Once the cluster labels are acquired for each selected base clustering pipeline, a counter encoder based on the implementation introduced in [105] transforms them into a numerical flavor. The cluster labels for each pipeline are replaced with their frequency, resulting in the encoded labels L_{en}. Similar to the dimensionality reduction that is used in building the data pipelines, we use Algorithm 3.5 as a second-stage encoding for the L_{en} to reduce the dimensionality into a lower space that maintains most of the original variance of the base clustering outcomes. Setting True to the u parameter applies a Kneedle method to determine the PCA components. If it is unattainable to accurately and precisely determine the knee index, the explained variance threshold is used instead, in which v is set to 95% to achieve the desired goal.

Selecting the Ideal Number of Clusters: Determining the optimal number of clusters is among the most challenging aspects of the clustering process. For example, the KMeans and hierarchical clustering algorithms require the number of clusters to be determined to group the data points accordingly. The distortion score helps identify the best number of clusters needed for clustering algorithms that require it. The score calculation involves determining the degree of divergence between each data point and its respective cluster center, with the result being the sum of squared distances. A higher score means the clustering outcome is less effective, while a lower score indicates the best effectiveness. Optimal cluster sizes can be determined by plotting the score against different cluster sizes and identifying the elbow point, the point at which the plot begins to level off. According to [106], the distortion score is mathematically expressed as:

$$D = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{15}$$

where k is the number of clusters, C_i is the set of data points allocated to cluster i, x data points $\in C_i$, and μ_i is the centroid of cluster i. The goal of clustering is to minimize the score as much as possible. Using this score in this study, we employed the knee point detection algorithm proposed in [91] to automatically identify the optimal elbow value for the number of clusters k, simplifying this critical process.

Table 3.5: Comparison of various pipelines using KMeans as a base clustering model.

(a) Data pipelines without PCA reduction

P	k	SC	CHI	DBI
P_1	4	0.7117	99388.6100	0.4755
P_2	5	0.2186	3620.2148	1.8650
P_3	5	0.3437	7154.0491	1.5209
P_4	5	0.3441	7162.4272	1.5148
P_5	4	0.3514	7669.0127	1.3448
P_6	5	0.6020	18893.5850	0.9479
P_7	5	0.2185	4133.8027	1.7617
P_8	5	0.3037	6794.3161	1.4906
P_9	5	0.3041	6726.1254	1.5256
P_{10}	5	0.2264	4206.1589	1.5352
P_{11}	4	0.4624	11827.3598	1.2293
P_{12}	5	0.2045	3627.9506	1.8595
P_{13}	4	0.4624	11827.3598	1.2293
P_{14}	4	0.4624	11827.3598	1.2293
P_{15}	5	0.3264	4040.4576	1.3211

(b) Data pipelines with PCA reduction

P	k	SC	CHI	DBI
P_1	4	0.7539	130024.7564	0.4165
P_2	4	0.2626	5793.9210	1.6221
P_3	5	0.4105	10377.3437	1.2641
P_4	5	0.4110	10401.4950	1.2573
P_5	4	0.4471	11913.9534	1.0607
P_6	5	0.6079	19117.8225	0.9420
P_7	5	0.2527	5186.1798	1.5283
P_8	5	0.3537	8986.5988	1.2605
P_9	5	0.3560	8869.1916	1.2949
P_{10}	5	0.2577	5349.1903	1.3634
P_{11}	4	0.5415	16482.6017	1.0515
P_{12}	5	0.2427	4635.1435	1.6448
P_{13}	4	0.5415	16482.6017	1.0515
P_{14}	4	0.5415	16482.6017	1.0515
P_{15}	5	0.4000	5080.3153	1.1497

Table 3.6: Comparison of various pipelines using Agglomerative as a base clustering model.

(a) Data pipelines without PCA reduction

P	k	SC	CHI	DBI
P_1	4	0.7101	98356.3247	0.4758
P_2	6	0.1891	3033.9224	1.8813
P_3	4	0.3747	7851.7142	1.4039
P_4	4	0.3746	7845.6106	1.4036
P_5	4	0.3037	6929.9167	1.5351
P_6	5	0.5753	16729.9033	1.0706
P_7	5	0.1953	3749.2588	1.8909
P_8	4	0.3373	7569.2856	1.4782
P_9	4	0.3403	7457.5812	1.4719
P_{10}	5	0.1630	3603.6385	1.7260
P_{11}	5	0.4942	11072.0702	1.1251
P_{12}	6	0.1779	2889.3170	1.8894
P_{13}	5	0.4942	11072.0702	1.1251
P_{14}	5	0.4942	11072.0702	1.1251
P_{15}	5	0.3278	3504.2283	1.4003

(b) Data pipelines with PCA reduction.

P	k	SC	CHI	DBI
P_1	4	0.7275	103705.2861	0.4620
P_2	5	0.2462	4701.0041	1.7023
P_3	5	0.4041	10154.6949	1.2841
P_4	4	0.4401	10905.5294	1.2039
P_5	4	0.3858	10287.4223	1.1625
P_6	4	0.5823	17582.1233	1.0915
P_7	5	0.2337	4799.6949	1.6786
P_8	4	0.3846	9646.3802	1.3170
P_9	4	0.3886	9505.3736	1.3100
P_{10}	5	0.1930	4501.7227	1.4896
P_{11}	5	0.5806	16557.4135	0.9216
P_{12}	6	0.2215	3904.0858	1.6708
P_{13}	5	0.5806	16557.4135	0.9216
P_{14}	5	0.5806	16557.4135	0.9216
P_{15}	5	0.3457	4202.0616	1.2989

3.2.2.3 Results and Discussion

Base Clustering Pipelines and Selection Results: We evaluated different data pipelines incorporating KMeans, Agglomerative, and MeanShift as base clustering models, analyzing their effectiveness with and without the implementation of PCA for dimensionality reduction, with setups as specified in Table 3.4.

The use of KMeans as a base clustering model, as shown in Table 3.5a and Table 3.5b, resulted in an overall enhancement of the structure of clusters with the application of PCA, as evidenced

Table 3.7: Comparison of various pipelines using MeanShift as a base clustering model.

(a) Data pipelines without PCA reduction

P	k	SC	CHI	DBI
P_1	4	0.7117	99382.9519	0.4756
P_2	1	0.0000	0.0000	0.0000
P_3	2	0.4027	10750.5777	1.1190
P_4	2	0.4021	10718.7755	1.1199
P_5	2	0.6520	8809.2248	0.5717
P_6	2	0.6390	27583.7521	0.6433
P_7	2	0.3412	7833.7724	1.3410
P_8	2	0.4066	11688.5227	1.0823
P_9	2	0.4011	11176.1000	1.0955
P_{10}	1	0.0000	0.0000	0.0000
P_{11}	1	0.0000	0.0000	0.0000
P_{12}	1	0.0000	0.0000	0.0000
P_{13}	1	0.0000	0.0000	0.0000
P_{14}	1	0.0000	0.0000	0.0000
P_{15}	11	0.3375	918.1624	1.2530

(b) Data pipelines with PCA reduction.

P	k	SC	CHI	DBI
P_1	4	0.7539	129966.3486	0.4165
P_2	1	0.0000	0.0000	0.0000
P_3	2	0.4426	13342.4127	0.9922
P_4	2	0.4420	13304.9449	0.9928
P_5	3	0.4553	12577.8194	0.9602
P_6	3	0.5849	18447.7587	1.2694
P_7	2	0.3685	9344.9197	1.2211
P_8	2	0.4383	13870.9523	0.9830
P_9	2	0.4321	13218.1198	0.9952
P_{10}	1	0.0000	0.0000	0.0000
P_{11}	3	0.5167	16090.8737	0.8242
P_{12}	1	0.0000	0.0000	0.0000
P_{13}	3	0.5167	16090.8737	0.8242
P_{14}	3	0.5167	16090.8737	0.8242
P_{15}	14	0.3356	914.3008	1.1252

Table 3.8: The selected clustering pipelines based on their combined scores: Results from Algorithm 3.4.

P	Base Model	k	SC'	CHI'	DBI'	CS
P_1	KMeans	4	100.0 %	100.0 %	75.53 %	91.93 %
P_1	MeanShift	4	100.0 %	99.96 %	75.53 %	91.91 %
P_1	Agglomerative	4	96.5 %	79.76 %	72.86 %	83.17 %
P_6	KMeans	5	80.63 %	14.7 %	44.66 %	47.01 %
P_{14}	Agglomerative	5	77.01 %	12.73 %	45.86 %	45.52 %
P_{13}	Agglomerative	5	77.01 %	12.73 %	45.86 %	45.52 %
P_{11}	Agglomerative	5	77.01 %	12.73 %	45.86 %	45.52 %
P_{13}	MeanShift	3	68.54 %	12.38 %	51.58 %	44.41 %
P_{11}	MeanShift	3	68.54 %	12.38 %	51.58 %	44.41 %
P_{14}	MeanShift	3	68.54 %	12.38 %	51.58 %	44.41 %
P_6	Agglomerative	4	77.24 %	13.52 %	35.88 %	42.56 %
P_{11}	KMeans	4	71.83 %	12.68 %	38.23 %	41.22 %
P_{13}	KMeans	4	71.83 %	12.68 %	38.23 %	41.22 %
P_{14}	KMeans	4	71.83 %	12.68 %	38.23 %	41.22 %
P_6	MeanShift	3	77.58 %	14.19 %	25.43 %	39.45 %
P_5	MeanShift	3	60.39 %	9.67 %	43.59 %	38.11 %
P_8	MeanShift	2	58.14 %	10.67 %	42.25 %	37.23 %
P_3	MeanShift	2	58.71 %	10.26 %	41.71 %	37.11 %
P_4	MeanShift	2	58.63 %	10.23 %	41.68 %	37.06 %
P_9	MeanShift	2	57.32 %	10.17 %	41.54 %	36.55 %
P_5	KMeans	4	59.3 %	9.16 %	37.69 %	35.63 %

by the improved SC. In particular, P_1 showcased the most substantial improvement, from 0.7117 without PCA to 0.7539 with PCA. Furthermore, the CHI increased across several pipelines when PCA is used, suggesting denser and more separated outcomes. For instance, P_1's CHI value rose

from 99388.6100 without PCA to 130024.7564 with PCA. However, some pipelines such as P_2, P_7, and P_{12} showed only minor improvements. The DBI, which has lower values indicating better clustering, generally decreased with PCA, signifying improved cluster compactness and separation. The most significant improvement is in P_1, as its DBI value decreased from 0.4755 to 0.4165.

In comparing Agglomerative clustering pipelines, we observed consistent improvements across all evaluation metrics when PCA is applied. As shown in Table 3.6a and Table 3.6b, there is an upward trend in the SC values with the application of PCA, indicating more coherent clusters. For instance, in P_1, the SC rose from 0.7101 to 0.7275, improving cluster quality outcomes. The CHI values also saw a general increase, suggesting that the clusters are better separated and denser with the implementation of PCA. Specifically, in pipeline P_1, the CHI increased from 98356.3247 to 103705.2861. Moreover, the DBI values generally decreased with PCA, indicating an enhancement in cluster compactness and separation. For example, the DBI in P_1 decreased from 0.4758 to 0.4620, showing a better-defined cluster structure.

By comparing various data pipelines using MeanShift clustering as a base model, we can observe a similar trend to KMeans and Agglomerative clustering pipelines, with PCA generally improving clustering performance across all evaluation metrics. As shown in Table 3.7a and Table 3.7b, pipelines with PCA show higher SC values, indicating that the PCA-reduced data produced more distinct and compact clusters. For instance, in P_1, the SC increased from 0.7117 to 0.7539. The CHI values also generally increased, indicating that clusters are better separated and denser when the data underwent PCA. It is particularly noticeable in pipeline P_1, where the CHI increased from 99382.9519 to 129966.3486 after PCA reduction. The DBI values generally showed a decrease with PCA reduction, demonstrating an improvement in the compactness and separation of clusters. In P_1, the DBI decreased from 0.4756 to 0.4165.

However, not all clustering pipelines benefited from PCA reduction and transformation using the MeanShift algorithm based on the default sklearn implementation, which estimates the bandwidth parameter that dictates the region's size to search through based on a heuristic technique. Pipelines P_2, P_{10}, and P_{12} are unable to find meaningful clusters (k=1), with SC, CHI, and DBI all at 0.0. In the case of P_{15}, it is observed that the number of clusters increased from 11 to 14 after applying PCA. This finding could complicate the interpretation of the results and render them less desirable

for specific applications.

Although PCA improved the overall clustering performance in many instances, it is worth noting that not all pipelines saw enhancement, especially when the original data structure is not well suited for linear transformation. The performance largely depends on the specific nature of the data transformation and normalization methods used within each pipeline. Thus, applying PCA should be considered in conjunction with these factors for optimal performance. Accordingly, the base clustering pipelines that incorporate applying PCA are scored and evaluated using Algorithm 3.4 - *Base Clustering Pipeline Selection*. This algorithm ranks and selects the best clustering pipelines based on their combined score of multiple validity indices representing their clustering performance. The output is a subset CP' from the initial grouping of base clustering pipelines, consisting of the highest-performing pipelines chosen for the final ensemble clustering solution.

Table 3.8 shows the selected clustering pipelines CP', sorted by their combined scores. These pipelines are selected based on a threshold set to the median of the combined score vector. Here, the combined score measures the overall performance of each pipeline considering all three normalized metrics (SC', CHI', and DBI'). As seen in the table, the pipeline P_1 with KMeans as the base model has the highest combined score and thus is the top selected model for the meta-clustering in the ensemble process. It is closely followed by the pipeline P_1 with MeanShift and Agglomerative as the base models, which also have high combined scores. On the lower end of the selection spectrum, we see that the pipelines using MeanShift on P_8, P_3, P_4, and P_9, and KMeans on P_5 have combined scores around 35-37%, still making the cut threshold for the ensemble process. This table represents the most promising combination of preprocessing and clustering algorithms for the final ensemble process. The aim is to combine these selected pipelines to leverage their strengths, thus creating an ensemble model with potentially better performance than any single model.

Counter Encoder-based Ensemble Clustering Results: The efficacy of the proposed ensemble clustering approach is evaluated using the selected clustering pipelines CP', shown in Table 3.8, employing several meta-clustering models, including KMeans, Agglomerative, and MeanShift algorithms. Our approach's effectiveness is evaluated using three clustering metrics (SC, CHI, and DBI) and with and without implementing dimensionality reduction based on PCA to the base clustering labels as a second encoding stage.

Table 3.9: Comparative performance of various meta-models with the proposed ensemble clustering: A juxtaposition of outcomes from base clustering and original data.

	Comparison with	**Meta Model**	k	**PCA**	**SC**	**CHI**	**DBI**
(a)	Respect to the Base Clustering Outcomes	KMeans	4	False	0.7297	25159.6013	0.8914
		KMeans	4	True	0.7371	27000.0676	0.8669
		Agglomerative	4	False	0.7223	23192.8589	0.8567
		Agglomerative	4	True	0.7171	23115.8426	0.7872
		MeanShift	64	False	0.9254	44697.3547	**0.1536**
		MeanShift	44	True	**0.9283**	**61871.5461**	0.3304
(b)	Respect to the Original Data	KMeans	4	False	0.7605	102978.2505	4.4805
		KMeans	4	True	0.7605	102978.2505	4.4805
		Agglomerative	4	False	0.7762	148892.2541	12.7522
		Agglomerative	4	True	**0.7877**	**188882.5009**	**2.8508**
		MeanShift	64	False	0.0793	22850.4584	96.6447
		MeanShift	44	True	0.2371	29348.1181	217.7144

For clarity, it is imperative to note that the metrics presented here have been intentionally presented in their unnormalized form. This decision is based on the rationale that a direct and unambiguous juxtaposition of raw scores across distinct models would be more illuminating. The proposed normalization methodology is meticulously formulated to serve the nuanced requirements of the clustering pipeline selection context rather than the general application.

Table 3.9a details the results of our proposed ensemble clustering approach concerning the base clustering outcomes. As we can see, each ensemble model version performs differently in terms of the used evaluation metrics. The MeanShift meta model with PCA produces the highest SC (0.9283) and CHI (61871.5461), indicating high separation and compactness among the clusters. Also, its DBI (0.3304), though not the lowest, is relatively low, which means there is less overlap between clusters. This finding suggests that the MeanShift meta model with PCA and 44 clusters yields the best overall performance considering the base clustering outcomes. However, it is essential to note that MeanShift results in a relatively high number of clusters (k) compared to KMeans and Agglomerative models. This case can increase the complexity of interpretation and may only be appropriate for some use cases. By utilizing the KMeans and Agglomerative models, we can achieve fewer clusters and fairly excellent scores, even without implementing dimensionality reduction. This outcome makes them a suitable and more accessible option for interpretation.

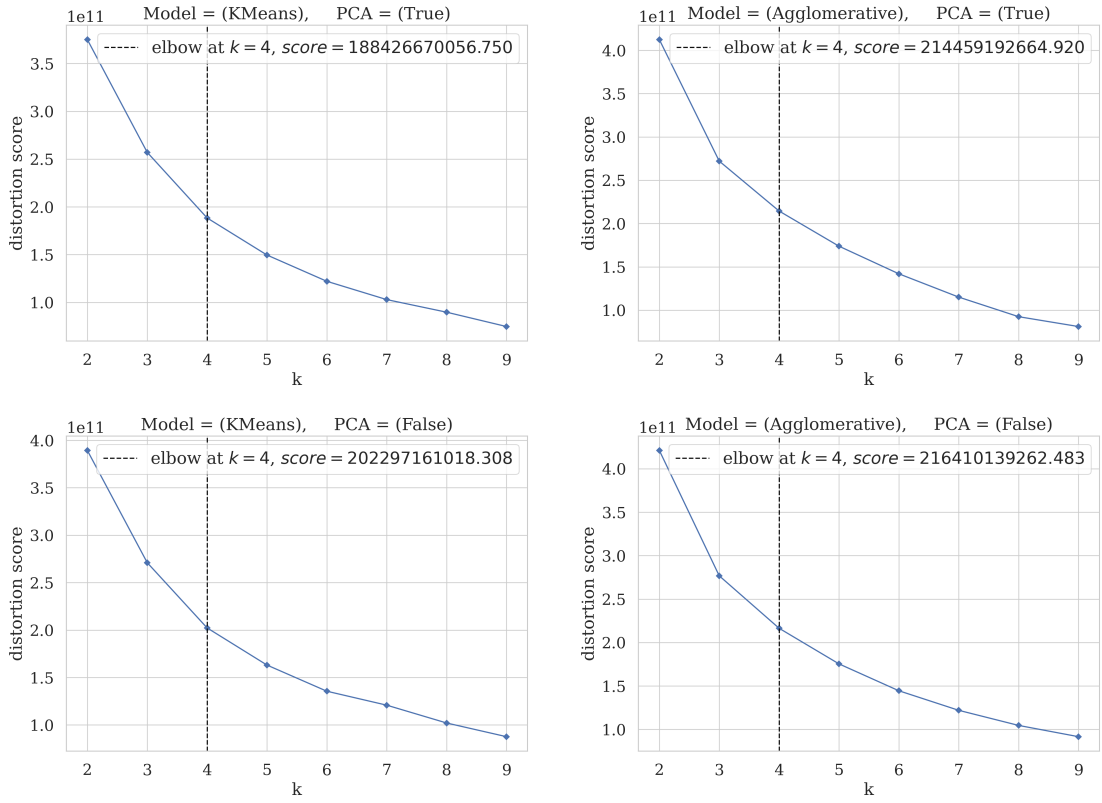

Figure 3.11: K-Elbow plots display the distortion score and fitting time in seconds for KMeans and Agglomerative clustering methods with and without applying dimensionality reduction using PCA.

In Table 3.9b, we present the results of our ensemble clustering approach, taking into account the original data projected into singular dimensions using PCA, as well as the clustering outcomes of all the meat clustering models with and without PCA consideration. The results show that using hierarchical clustering with PCA reduction resulted in the best performance. Specifically, it produced the highest SC (0.7877) and CHI (188882.5009), which indicates a high degree of separation and compactness among the clusters. The DBI (2.8508) is the lowest among all models, suggesting minimal overlap between the clusters. Meanwhile, using the MeanShift model shows considerably lower SC and CHI scores and notably higher DBI with and without PCA consideration. This case indicates that the MeanShift as a meta-clustering model produces less cohesive and more overlapping clusters than the other models due to the highest cluster numbers.

Interestingly, the PCA and non-PCA variants of the KMeans model yield identical results concerning the projected original data into a singular dimension. It suggests a robust inherent structure

in the data, where PCA's dimensionality reduction does not significantly alter the primary clustering patterns. Such congruence implies that the clusters identified are defined by substantial structural differences that remain consistent irrespective of the PCA transformation. The stability in clustering outcomes across both variants underscores the strong, intrinsic data patterns and the KMeans model's sensitivity to these overarching patterns.

Nevertheless, each model has distinct strengths and limitations. The selection between them should be judiciously based on the intricacies of the data set in use and the overarching goals of the clustering endeavor. For instance, in a data set where the first few principal components primarily capture the variance, the PCA variant might be more efficient by reducing computational costs without sacrificing clustering accuracy. Conversely, for a data set where vital information is dispersed across multiple components, the non-PCA version might be better suited to capture the slight difference in the data. Nevertheless, such a form will increase the computational costs.

However, in Figure 3.11, we present the elbow analysis for KMeans and Agglomerative meta clustering models, both with and without dimensionality reduction. The graph displays the distortion score for various cluster numbers (k) ranging from 2 to 10. Importantly, the chosen k is shown as a dashed line for each model.

Figure 3.12 visually represents the intercluster distance in a two-dimensional space. This figure showcases the outcomes of KMeans and MeanShift meta-clustering models, both with and without dimensionality reduction. The feature space is embedded using a Multi-Dimensional Scaling (MDS) algorithm presented in [106]. It aims to transform the base clustering outcomes into a low-dimensional space while preserving the pairwise distances between the original points as much as possible. Correspondingly, it is paramount to underscore that the overlapping of two clusters in the 2D space does not inherently signify an overlap in the original feature space. The clusters' size indicates membership, allowing for a visual gauge of the relative importance of each cluster. This visual helps to identify the important clusters and facilitates decisions based on their characteristics.

Lastly, Figure 3.13 displays silhouette analysis plots for KMeans meta clustering both with and without the application of dimensionality reduction. The dashed line in the figure indicates the average scores, which are observed to be nearly equal. It is noteworthy to mention that despite the dimensionality reduction, the clustering results of both versions are congruent. This consistency

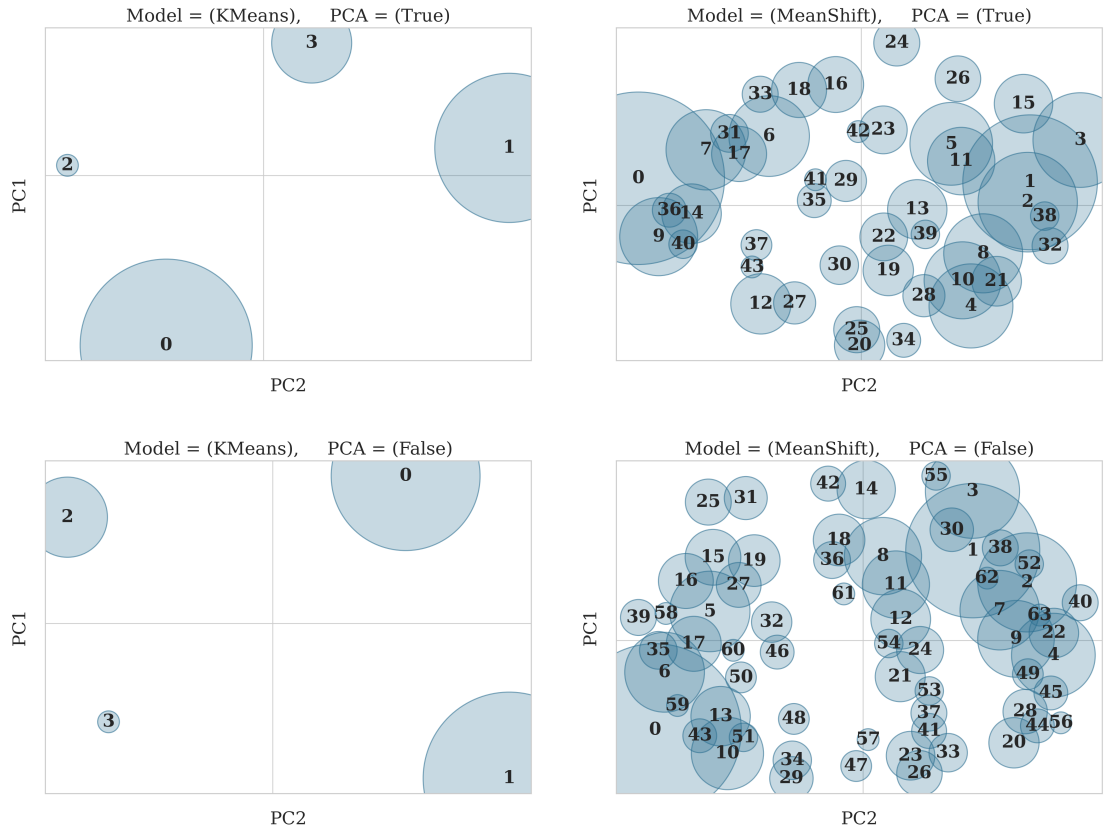

Figure 3.12: Intercluster distance map for KMeans and MeanShift clustering methods with and without implementing dimensionality reduction, embedded via the MDS.

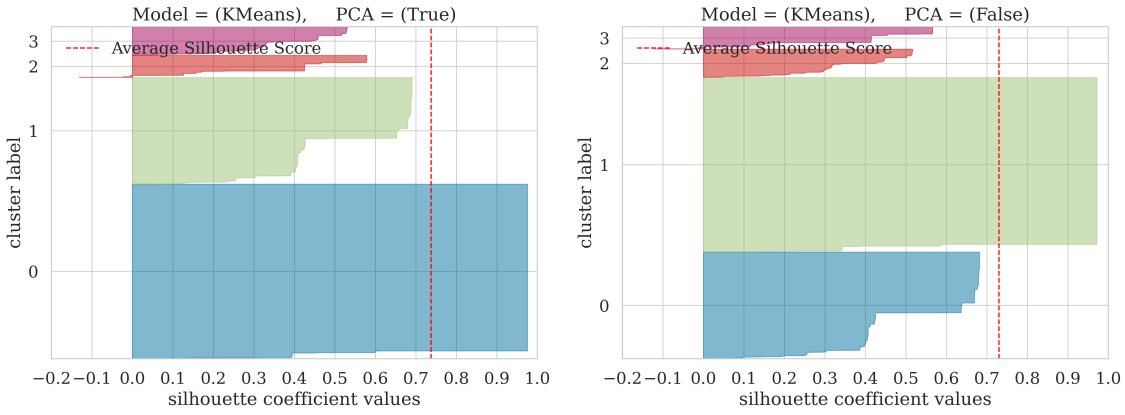

Figure 3.13: Silhouette analysis plot of KMeans clustering method with and without implementing dimensionality reduction.

underpins the robustness of the underlying data structure and the model's ability to discern intrinsic patterns, regardless of the application of dimensionality reduction.

3.2.3 Conclusion

In this study, we have introduced a novel ensemble clustering technique for categorizing cloud workloads. The proposed approach, coupling multiple data preprocessing pipelines with diverse base clustering learners, has demonstrated remarkable potential for uncovering and capturing complex categorization perspectives. Through rigorous testing using real-world trace data from Microsoft Azure, we have provided empirical evidence of the effectiveness of our approach. A unique combined scoring method is proposed to select the most influential models and preprocessing setups, providing valuable insights into the performance of various combinations of clustering algorithms and data preprocessing techniques. Our findings pave the way for a new perspective in managing cloud resources, whereby an advanced ensemble clustering approach can effectively navigate the multifaceted nature of cloud workloads.

This study has been published in [107], and our future research is to explore additional ways to enhance this approach, incorporating additional machine learning techniques and fine-tuning the preprocessing pipelines to accommodate the evolving nature of cloud workloads. Additionally, investigating the applicability of our approach in different cloud environments beyond Microsoft Azure may provide further insight into its universal applicability and robustness. As cloud computing grows in complexity and scale, it becomes crucial to discover the latent categorization perspectives inherent in cloud data center workloads. By doing so, we can gain deeper insight, facilitating improved decision-making processes in resource allocation, performance optimization, and workload balancing within cloud data centers. It can also increase overall operational efficiency, translating into improved business results and customer experiences. Therefore, the methodology presented in this study provides a robust framework for optimizing future cloud resources.

Chapter 4

Enhanced Single-Output Predictive Modeling in Cloud Computing

In this chapter, we explore various methodologies and innovations in predictive modeling within the realm of cloud computing. This exploration is segmented into three pivotal sections, each addressing a unique aspect of predictive modeling in cloud environments. In the beginning, we explore an advanced multilevel learning model specifically created to forecast CPU utilization in cloud data centers, as described in Section.4.1. This model is crucial for providing high prediction precision by incorporating various machine learning models at three levels of learning. Next, the focus shifts to the economic aspects of cloud computing in Section 4.2. This section presents a regression-based approach for proactive predictive modeling, which is instrumental in navigating the complexities of a prediction-based cloud service pricing model. Finally, the chapter concludes with Section 4.3, where we explore the use of imbalance and ensemble learning methods. This section focuses on enhancing load prediction in cloud computing systems, a key factor for ensuring efficient resource management in dynamic cloud environments. Each section of this chapter collectively contributes to a deeper understanding of predictive modeling in cloud computing, showcasing the latest advancements and their practical applications.

4.1 A Multilevel Learning Model for Predicting CPU Utilization in Cloud Data Centers

In the contemporary era of cloud computing, efficient and precise prediction of CPU utilization ensures optimal performance and energy efficiency in data centers. Traditional predictive models often need to be improved as these data centers grow in complexity and scale, necessitating more nuanced and integrative solutions. This study introduces an advanced multilayered learning framework meticulously designed to meet the demands of modern cloud data centers. Our innovative approach synergistically combines anomaly detection, data clustering, and ensemble-based regression prediction. The Isolation Forest algorithm is used during the preliminary stage to identify and address anomalies within the data. Subsequent phases harness the KMeans clustering algorithm, refine data categorization based on recurrent CPU usage patterns, and employ multilevel ensemble-based prediction models for accurate forecasting rooted in historical and real-time data trends. Through comprehensive evaluations, our model demonstrates significant improvements in prediction accuracy and robustness against the dynamism inherent in cloud environments. Our research paves the way for a more resilient, proactive, and efficient approach to CPU utilization prediction, laying the foundational stone for future innovations in cloud computing resource management.

4.1.1 Initial data preparation

As shown in Figure 4.1, the model architecture involves acquiring time series data related to CPU utilization of hosts running multiple VCIs. A sliding windowing method is applied to the collected data as part of the initial data preparation. This process involves identifying the feature space derived from the host CPU utilization and the target value as the subsequent CPU utilization. These values are determined based on a prescribed window size (w) and a step size (s). In addition, we extract date-time indexes, such as month, day, hour, and minute, from each derived window timestamp, including information about the host, such as the count of hosts VCI and the available CPU and memory capacities. The gathered information is then used to build the ultimate feature space, which is used to train ML algorithms. In anomaly detection and clustering processes, the extracted feature space is ultimately reduced to a single component using PCA, a popular dimensionality

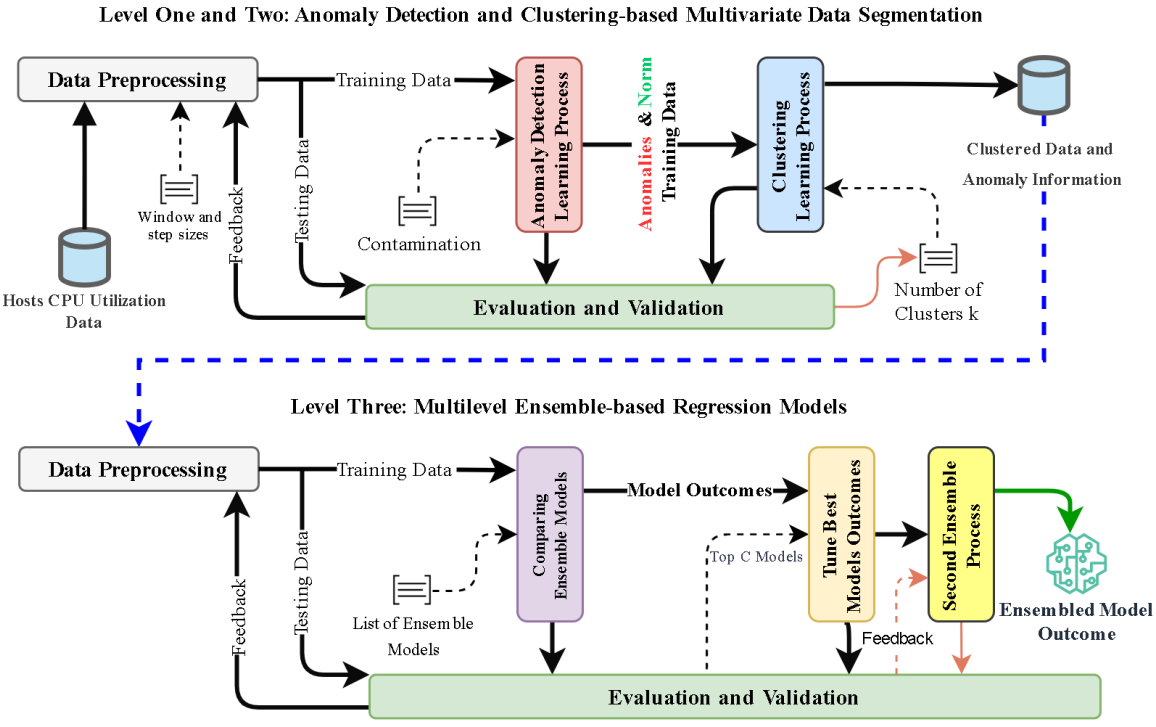

Figure 4.1: Proposed Model Architecture

reduction technique.

4.1.2 Detecting and identifying anomalies

Following the initial phase of data preparation, the next step in our framework is dedicated to detecting and identifying anomalies within the data set that can potentially negatively impact the precision of predictive models. To achieve this objective, we employ the Isolation Forest algorithm, which is highly regarded for its effectiveness in identifying and isolating anomalous data points. The algorithm under consideration is an unsupervised learning technique that uses a stochastic feature selection procedure, subsequently determining a split value within the range of minimum and maximum values of the chosen feature [108]. This process makes it inherently easier to isolate anomalies than regular observations. The contamination parameter, denoted as F, within the context of the isolation forest algorithm, represents the ratio of outliers present within a given data set. The parameter can be adjusted to refine the threshold limit distinguishing outliers from regular observations. On execution of the algorithm, an anomaly score is assigned to each data point, in which

instances that exceed the contamination threshold are classified as anomalies.

4.1.3 Data Clustering

Following anomaly detection, the next step involves grouping similar data points considering the anomaly information. We employ the KMeans clustering algorithm, a popular partition-based clustering technique [97]. The KMeans algorithm operates by initializing the centroids k in the data space, where k represents the predetermined number of clusters. During each iteration, data points are assigned to the nearest centroid. Afterward, the centroids are recalculated by taking the average of the data points that have been assigned. This process iterates until the centroids stabilize and no longer change significantly or until a predefined number of iterations is reached. One of the primary advantages of KMeans is its efficiency, especially with large datasets.

However, determining the optimal number of clusters poses a significant challenge in clustering. Utilization of distortion score is employed in order to determine the most suitable number of clusters. The score is computed by adding the squared distances between each data point and its corresponding cluster centroid. A higher clustering performance is associated with lower score values. By graphing the scores for various clusters, one can determine the optimal number of clusters that leads to the most substantial reduction in the distortion score. This location is known as the elbow point because it is where the plot typically begins to flatten. The distortion score can be mathematically defined as:

$$D = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{16}$$

where k is the number of clusters, C_i is the set of data points assigned to cluster i, x is a data point, and μ_i is the centroid of cluster i. Using this score, we employ a knee point detection algorithm proposed in [91] to identify the elbow point that represents the optimal k. This approach ensures a reliable and unbiased analysis. Once each data point is clustered, we calculate the Silhouette Coefficient (SC) score for each data point and transform the cluster labels using one-hot encoding. Section 4.1.5.2 presents more details about SC score.

4.1.4 Ensemble-based Prediction

Based on previous results, this phase uses multilevel ensemble-based regression models to predict future CPU utilization. Ensemble learning techniques leverage the principle that the combination of multiple models can often lead to improved performance over any individual model [109]. In the context of our research, this approach is particularly potent, given the dynamism and variability inherent in cloud data center CPU utilization patterns. Initially, we employ a series of baseline ensemble models, including popular methods such as random forests, gradient boost machines, and AdaBoost. Each of these models aggregates the predictions of several base estimators, usually decision trees, to produce a final forecast. These models are systematically trained and validated on our clustered data, and their performances are compared using a predefined metric such as the coefficient of determination.

Upon evaluation, the top models M that demonstrate the highest predictive accuracy and robustness against overfitting are selected for further ensemble process. To take our ensemble strategy a step further, these M models are integrated using advanced techniques such as stacking or voting regression. The stacking regression is operated by training a meta-model on the predictions of the selected M models [110]. Essentially, while the initial models make their predictions, the meta-model learns how to combine these predictions to yield a final, more refined output. The Voting Regressor, on the other hand, functions by taking an average or weighted average of the predictions of the selected models [111].

Following this second level of an ensemble, a rigorous evaluation is conducted to determine which method, Stacking or Voting, offers superior performance. The best model is then selected for deployment accordingly, ensuring that the CPU utilization predictions are accurate and robust, accounting for many potential scenarios in cloud data centers. The abstract steps of the proposed multilevel learning model for CPU predictions are outlined in Algorithm 0.

However, it should be noted that ensemble techniques can increase the accuracy of the prediction and introduce additional complexity. Therefore, a careful balance between model performance and interpretability is maintained throughout our methodology. In culmination, our multifaceted approach delivers precise, actionable insights, drawing from the strengths of each component to

Algorithm 4.1 Multilevel Learning Model for CPU Predictions

Require: Time series data (Hosts CPU utilization)
Require: Window size w, Step size s
Require: Contamination parameter F
Require: Predefined number of iterations mi for KMeans
Require: Baseline ensemble models B
Require: Meta-model m for stacking
Ensure: Best ensemble model

1: Collect hosts data
2: Apply sliding window method with size w and step s
3: Extract date-time indexes and host information
4: Reduce the feature space using PCA for anomaly detection and clustering processes
5: Use Isolation Forest with contamination parameter F to detect anomalies
6: Assign an anomaly score to each data point
7: Classify instances that exceed the contamination threshold as anomalies
8: Determine the optimal k using the distortion score (Equation 16) and the knee point detection algorithm [91]
9: Group the data using KMeans clustering with initialized centroids k and max iterations mi
10: Calculate the silhouette coefficient score for each data point using Equation 1
11: Transform the cluster labels using the one-hot encoding
12: Train baseline ensemble models B
13: Select the top models M based on predictive precision and robustness using Equation 20
14: Combine the M models by utilizing a Stacking Regressor based on the meta-model m
15: Combine the M models by utilizing a Voting Regressor
16: Evaluate the performance of Stacking vs. Voting Regressors and determine the best model using Equation 20
return Best ensemble model

address the multifaceted challenges posed by modern cloud data centers.

4.1.5 Experimental Setup and Evaluation

The evaluation of our proposed model involves using a real workload and various metrics to assess the effectiveness of anomaly detection, data clustering, and ensemble prediction processes. More details about the evaluation metrics, experimental setup, and results are given below.

4.1.5.1 Evaluation for Anomaly Detection

In the absence of ground-truth labels, evaluating the results of unsupervised anomaly detection requires an alternative approach. A prominent technique that we used is Kernel Density Estimation (KDE) [112]. The KDE is a nonparametric technique that can provide insight into anomaly

distribution by estimating the probability density function of continuous variables (which have been reduced to a single component using PCA in our assessment). By plotting KDE, we display the density of the anomaly, facilitating the intuitive identification of potential threshold values that distinguish between anomalous and regular data points.

4.1.5.2 Evaluation Metrics for Clustering

Similarly to anomaly detection, we use KDE to show the distribution of data clusters by estimating the probability density function of the reduced data to a single component using PCA. Furthermore, the performance of the clustering is examined using different intrinsic metrics such as SC [92], CHI [93], and DBI [94], which are explained in Section 3.1.2.1 of the previous Chapter 3.

4.1.5.3 Evaluation Metrics for Regression-based Prediction

A set of metrics is used to evaluate the performance of the regression-based ensemble predictions. As outlined in [113], each metric provides a unique prediction accuracy and perspective on error dispersion. Assume that $\hat{y}_i$ is a predicted value for i-th point, y_i its actual value and n is the dataset size; below are more details of each metric comprising its respective equation.

Mean Absolute Error (MAE) calculates the mean of the absolute differences between the predicted and actual values. It assigns a linear penalty for each unit of the discrepancy between the predicted and observed values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{17}$$

Mean Squared Error (MSE) computes the average squared difference between the predicted and actual values. It gives a higher penalty for larger errors, making it more sensitive to outliers than the MAE metric.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{18}$$

Root Mean Squared Error (RMSE) is the square root of MSE. RMSE has the benefit of punishing larger errors more severely like MSE. Additionally, it is measured in the same unit as the

dependent variable.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{19}$$

Coefficient of Determination (R^2) metric indicates the model's goodness of fit. It represents the proportion of variance in the dependent variable that is predictable from the independent variables. An R^2 of 1 indicates perfect prediction, while an R^2 of 0 indicates that the model does not improve the prediction over the mean of the target.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{20}$$

Mean Absolute Percentage Error (MAPE) is a relative measure of the accuracy, which calculates the average percentage error between the predicted and actual values.

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \tag{21}$$

4.1.5.4 Experimental Setup

The core of our experimental data is sourced from the CoMon project, which captures the workload of more than 1,000 VMs running on 800 hosts over two days. We used the CloudSim simulation tool to execute the workload, focusing on adherence to a specific VM selection policy (focusing on minimizing migration time) and a VM allocation policy (based on a static threshold). These policies are proposed in [114], based on which the host's CPU utilization is extracted. The initial preprocessing step involves using the sliding window to simplify the subsequent analysis. It is done using a window of one hour (window size w) and advancing in 5-minute intervals (step size s).

For the anomaly detection process, we used the Isolation Forest algorithm with 100 base estimators and the contamination parameter F calibrated to 0.05. For the clustering process, we used the KMeans algorithm with maximum iterations mi set to 300 to guarantee the computational efficiency and dependability of the results. We used multiple baseline ensemble models B for the ensemble-based prediction process. These encompassed: Gradient Boosting Regressor, Random Forest Regressor, Extra Trees Regressor, CatBoost Regressor, Light Gradient Boosting Machine,

Extreme Gradient Boosting, and AdaBoost Regressor. The best model threshold M is set to 4, selected for the second ensemble process based on R^2. All these models, including the Voting and Stacking regressors (with RidgeCV as meta-model m), are initialized and operated using the default settings provided by the `scikit-learn` [9], and time series cross-validatora with three folds.

4.1.5.5 Evaluation Results

Anomaly detection results The graph in Figure 4.2 shows the KDE distribution of the first principal component for both normal and abnormal classes in the training and testing datasets. It provides a smoothed representation of the distribution of data points, with the first principal component (PCA0) on the x-axis and the density of occurrences on the y-axis. By distinguishing the data based on their class labels (1 anomaly or 0 norm), the graph illustrates how each data class is spread across the principal component.

However, the graph in the training and testing datasets of the KDE plots reveals distinct modes and peaks. As observed, a prominent peak represents normal data, while secondary peaks and extended tails signify anomalies. Additionally, the trough between these peaks serves as a natural threshold for classification, proving to be particularly helpful for displaying anomaly score distributions after reducing the dimensions of the data set to one. In general, these graphs present a valuable visual representation of the distribution of the data in reduced dimensions, helping to identify patterns, separations, and overlaps that influence the analysis and modeling decisions.

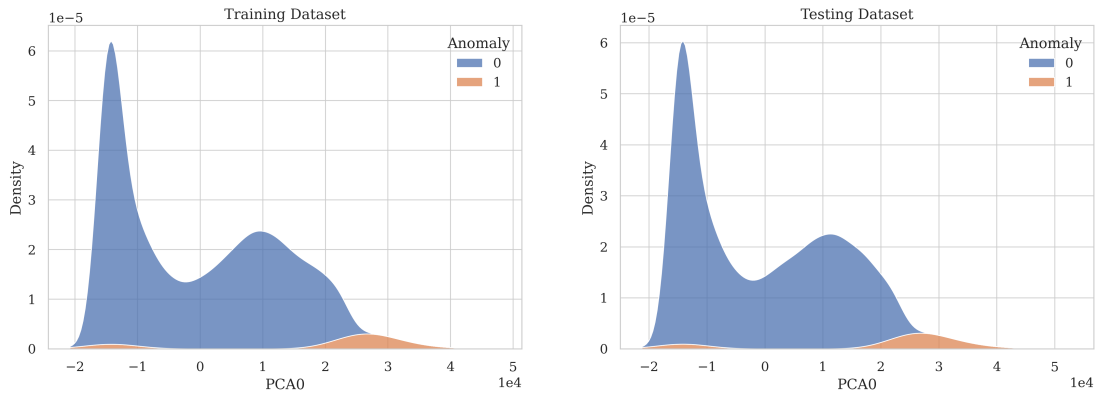

Figure 4.2: KDE distribution plots of the first principal component for normal and anomalous classes in training and testing datasets

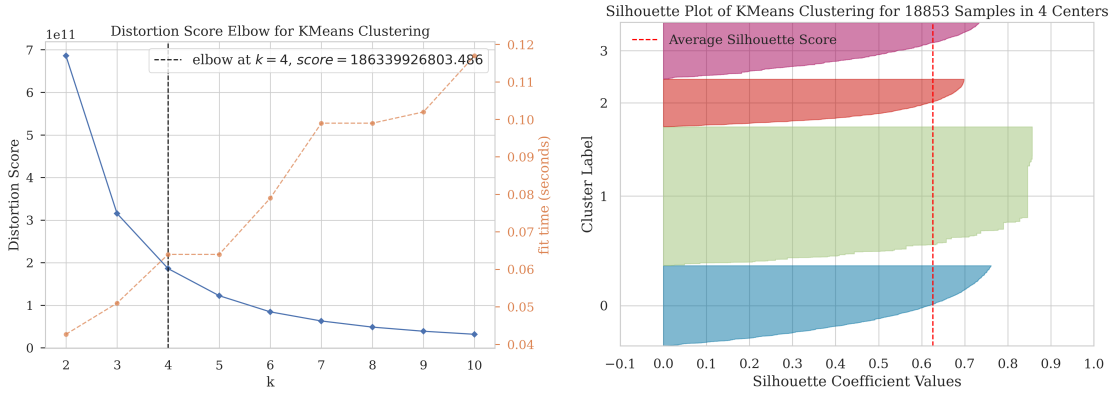

Figure 4.3: KElbow and silhouette analysis plots of KMeans clustering method

Data clustering results Figure 4.3 shows the KElbow and silhouette coefficient analyses of the KMeans clustering method. The point at the elbow indicates the most suitable number of clusters observed at ($k = 4$). It suggests that increasing the number of clusters beyond four would not significantly reduce the within-cluster sum of squares; therefore, four is the optimal number of clusters for the given data set. However, the silhouette coefficient analysis provides information about the separation distance between the resulting clusters. Higher average silhouette coefficients indicate that clusters are well separated. As observed, the mean SC for the clustering solution with (k = 4) is 0.6256, which is a reasonably high score. It indicates that the data points in each of the four clusters are, on average, closer to other data points in their cluster and farther away from the data points in other clusters. In addition, CHI is 106520.0213, which is a high value indicating that the clusters are dense and well separated, which means that the KMeans algorithm has partitioned the data into potentially meaningful clusters. The DBI is recorded at 0.5044. This value is closer to 0, which indicates better partitioning, and the low value here suggests that the clusters generated by the KMeans algorithm are distinctly separated from each other.

In general, the metrics presented point towards an effective clustering of the data by the KMeans algorithm, characterized by well-defined, well-separated, and not overlapping clusters. Figure 4.4 shows the KDE distribution graphs of the first principal component for grouping classes in the training and testing datasets, which confirms this finding.

Ensemble-based prediction results Table 4.1 presents a comparative analysis of the performance of various baseline ensemble models used in the ensemble-based prediction. Seven distinct

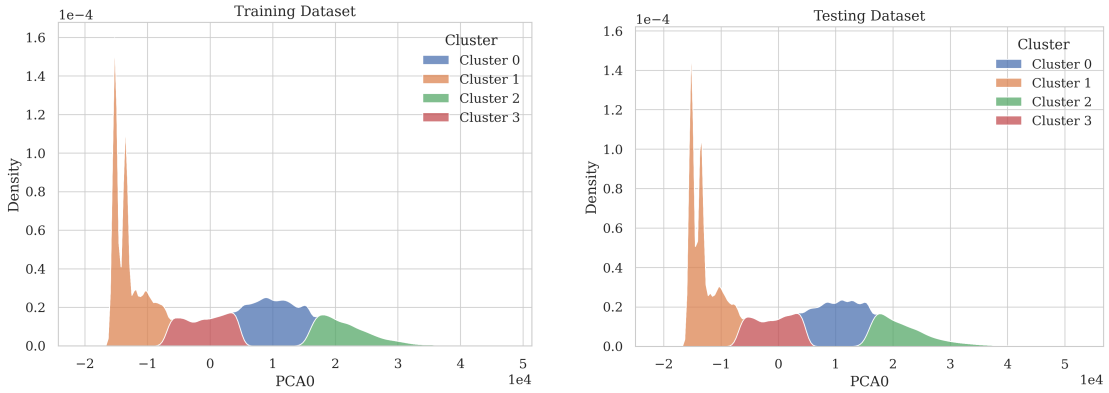

Figure 4.4: KDE distribution plots of the first principal component for clustering classes in training and testing datasets

Table 4.1: Comparison results of baseline ensemble models

Model	MAE	MSE	RMSE	R^2	MAPE
Gradient Boosting	0.0509	0.0058	0.0758	0.9450	21.4249
Random Forest	0.0516	0.0060	0.0772	0.9429	17.8271
Extra Trees	0.0518	0.0060	0.0773	0.9428	19.2034
CatBoost	0.0517	0.0060	0.0776	0.9424	17.9577
Light Gradient Boosting	0.0518	0.0060	0.0777	0.9422	19.5736
Extreme Gradient Boosting	0.0550	0.0068	0.0826	0.9347	18.1816
AdaBoost	0.0718	0.0083	0.0909	0.9209	103.1659

models are evaluated using multiple metrics. The results show that all models performed relatively accurately, as indicated by their respective R^2 values. The Gradient Boosting Regressor emerges as the top performer with a R^2 value of 0.9450, which suggests that the model explains approximately 94.50% of the variability in the target variable. It is closely followed by the Random Forest Regressor and the Extra Trees Regressor with R^2 values of 0.9429 and 0.9428, respectively.

In order to accurately assess the precision of the model's prediction, it is essential to take into account various error metrics. For instance, in terms of MAE, the Gradient Boosting Regressor has the slightest error, denoting its superiority in terms of absolute deviations from the actual values. The same model also performed commendably in terms of RMSE and MSE. Interestingly, although the AdaBoost Regressor has a respectable R^2 value of 0.9209, it shows significantly higher MAE and MAPE, suggesting larger deviations in predictions on an absolute scale and percentage-wise. Another crucial factor to consider, especially for real-time applications or large datasets, is the time

Table 4.2: Cross-validation scores by fold for Stacking Regressor

Fold	MAE	MSE	RMSE	R^2	MAPE
0	0.0515	0.0059	0.0769	0.9440	21.9566
1	0.0504	0.0056	0.0748	0.9460	18.2230
2	0.0497	0.0057	0.0753	0.9455	18.1845
Mean	0.0505	0.0057	0.0757	0.9452	19.4547
Std	0.0007	0.0001	0.0009	0.0009	1.7692

Table 4.3: Cross-validation scores by fold for Voting Regressor

Fold	MAE	MSE	RMSE	R^2	MAPE
0	0.0517	0.0060	0.0773	0.9435	20.6737
1	0.0506	0.0056	0.0751	0.9455	18.5245
2	0.0501	0.0057	0.0758	0.9448	18.0903
Mean	0.0508	0.0058	0.0761	0.9446	19.0961
Std	0.0006	0.0001	0.0009	0.0009	1.1295

Table 4.4: Performance Results of Stacking Regressor versus Voting Regressor on a Test Dataset

Model	MAE	MSE	RMSE	R^2	MAPE
Stacking Regressor	0.0512	0.0061	0.0783	0.9409	34.9491
Voting Regressor	0.0513	0.0062	0.0788	0.9401	27.4202

taken for training. Extreme Gradient Boosting, with its remarkable efficiency, took the least time of 0.2533 seconds, even though it had to compromise slightly on some error metrics.

In the context of the ensemble-based prediction process mentioned, the top four models, based on R^2, that would be selected for the second ensemble process are Gradient Boosting Regressor, Random Forest Regressor, Extra Trees Regressor, and CatBoost Regressor. This ensemble process combines multiple models' predictions to obtain more accurate results. As we explained earlier, stacking and voting regressors are two popular techniques used in this process. Stacking involves combining the outputs of several models and using them as input to a meta-model. The meta-model then learns how to combine the predictions of the base models to achieve better performance. Voting, on the other hand, involves combining the predictions of the base models by taking a simple majority vote. The stacking and voting regressors are used as a second level of the ensemble process to further improve the accuracy of the predictions.

Table 4.2 presents the cross-validation results of the Stacking Regressor over three folds. The

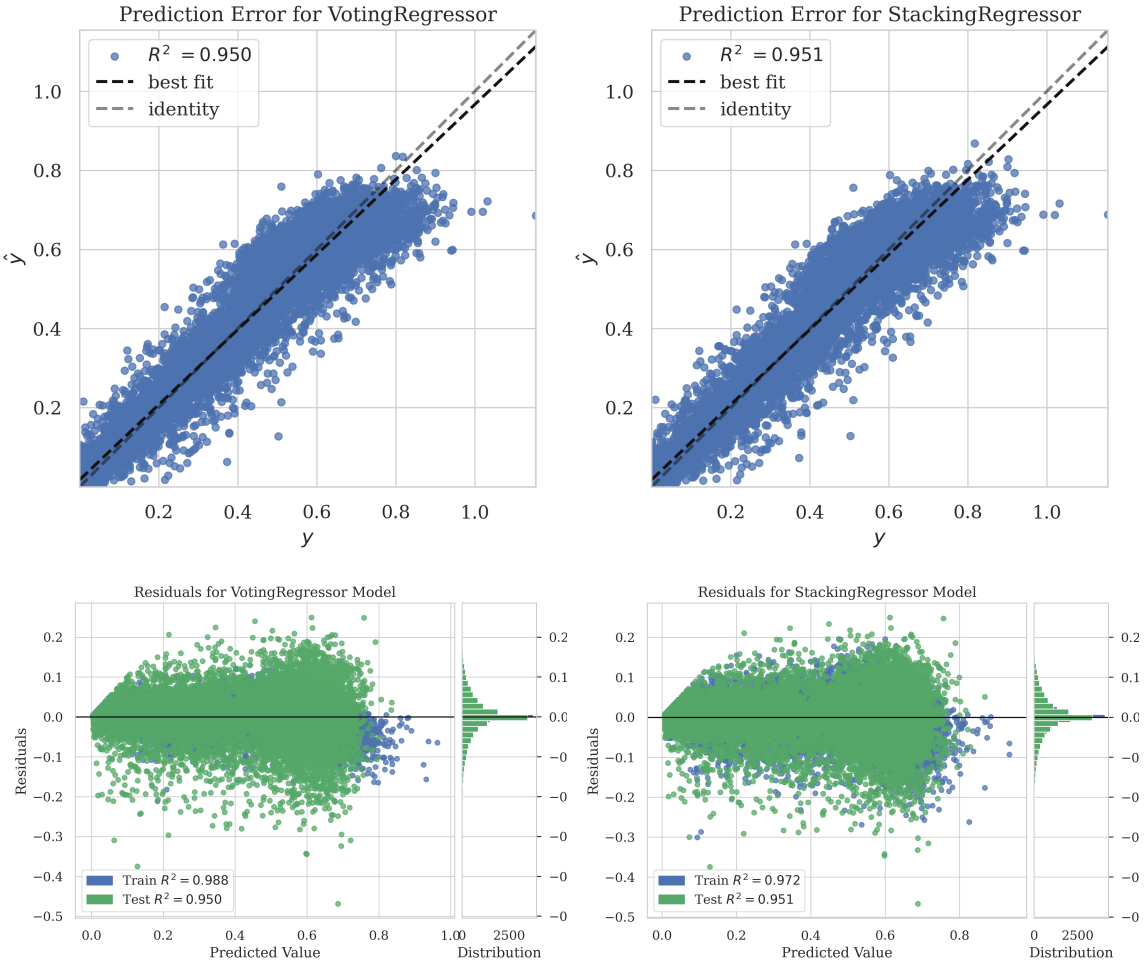

Figure 4.5: Residuals and prediction error plots for the Stacking and Voting Regressors without the consideration of the cross-validation process

mean R^2 value stands at 0.9452, indicating that the model explains approximately 94.52% of the variance in the data set. Similarly, Table 4.3 presents the cross-validation metrics for the Voting Regressor. It shows a comparable mean R^2 of 0.9446. Although both ensemble strategies appear to have performed closely in cross-validation, it is crucial to note the performance on an unseen test data set for a more holistic understanding of their robustness. Table 4.4 presents the performance of the Stacking and Voting Regressors on a test data set. Accordingly, the Stacking Regressor will be selected for deployment, with a slightly higher R^2 of 0.9409 compared to the Voting Regressor's 0.9401, indicating a marginal superiority. However, the Voting Regressor exhibits a somewhat lower MAPE, which might indicate better general performance in specific applications.

In Figure 4.5, we investigate the residuals and prediction errors of the Stacking and Voting Regressors, bypassing the lens of the cross-validation process. The Stacking Regressor demonstrates consistent performance with an R^2 of 0.972 on the training set and 0.951 on the test set. On the other hand, the Voting Regressor boasts an R^2 of 0.988 for the training data but slightly trails the Stacking Regressor on the test set with 0.950. These results suggest that while both models are highly effective, there is a subtle trade-off between fitting the training data and generalizing it to new data. The choice between these ensemble strategies can be application-specific and depends on the trade-offs one is willing to make in terms of predictive accuracy, interpretability, or computational efficiency.

4.1.6 Conclusion

In the ever-evolving landscape of cloud computing, ensuring efficient CPU utilization remains a pressing concern. This study introduced a novel approach that combines anomaly detection, clustering, and multilevel ensemble-based predictions to predict CPU usage. At the heart of our solution lies the harmonious integration of three distinctive techniques. The Isolation Forest algorithm, adept at recognizing outliers, ensures robustness. Applying the KMeans clustering algorithm refines the data, increasing the prediction granularity. Finally, a multilevel ensemble prediction, which harnesses foundational ensemble models and advanced meta-ensembling techniques like Stacking and Voting regressors, ensures that the forecasts are precise and adaptable to the multifaceted dynamics of cloud data centers. This study is published in [115] and sets the foundation for improving cloud resource management and can guide future research on predictive analytics in cloud computing. Future work could improve our architecture using hybrid models and real-time adaptation.

4.2 Regression-Based Approach for Proactive Predictive Modeling of Efficient Cloud Cost Estimation

Businesses increasingly lean on cloud-based solutions in the contemporary digital landscape, drawn by their scalability and adaptability. Navigating the financial intricacies of cloud subscription models, particularly when intertwined with software-defined systems, remains a formidable

challenge. This challenge is accentuated by dynamic pricing structures, making accurate cost forecasting a critical but complex endeavor. This research unveils an innovative methodology designed to meticulously forecast the financial costs of cloud subscription tasks based on resource allocation characteristics. Our approach centers around a robust model carefully created through particular preparation and modeling stages that incorporate a pricing model for various virtual machines and utilize different advanced regression algorithms to navigate the complex world of cloud subscription services. The results of our study, which analyzed real cloud workloads, indicate that equipping businesses with a powerful predictive tool can lead to improved financial planning and strategic decision-making in their cloud operations. This study aims to guide businesses in the ever-changing field of cloud technology, focusing on promoting financial efficiency and improving decision-making strategies in their cloud initiatives.

4.2.1 Model Design

The architecture of the proposed model, as illustrated in Figure 4.6, provides a comprehensive representation of its structure and emphasizes its systematic methodology to forecast cloud subscription costs through a regression model. Each model segment's design has been subjected to detailed design considerations, utilizing advanced methodologies and iterative refinement processes to create a robust and reliable system.

The initial phase of the model workflow is the extraction stage. The model gleans VMs traces from various cloud subscriptions. The specificity of these traces, defined by a set temporal window, ensures that the data capture the intricacies and evolving trends characteristic of the selected time frame. Such precision is vital, as it directly influences subsequent stages and the overarching fidelity of the predictions. After data extraction, the model delves into a comprehensive data preparation stage, explored in detail in Section 4.2.2. An essential procedure in this phase involves calculating the aggregate price associated with each subscription. This calculation is not trivial; it requires integrating a VM pricing model, as described in Section 4.2.3.

The model embarks on its evaluative journey once the data have been assimilated and adequately prepared. The vast realm of ML offers a myriad of regression algorithms, each with distinct advantages and limitations. In this model, a set of regression candidates is subjected to rigorous testing

and evaluated against performance benchmarks. The objective is not solely to discover a model that exhibits high performance on the training data set but also to ascertain a model that can effectively generalize to unseen data. This procedure is essential to build a reliable ML model.

At the end of the evaluation phase, when a superior regression model has been identified, it is subjected to a registration process. Selection and activation of the desired model play a crucial role in enabling real-time predictions, facilitating interaction with live data streams with Software-Defined Systems (SDS) interfaces, and providing valuable feedback information. However, the journey of the model does not conclude post-deployment. Given the inherently dynamic nature of the cloud computing sphere, it is imperative that the model continuously adapts. As such, ingrained within the model's architecture are mechanisms dedicated to its ongoing monitoring. These periodic evaluations ensure its performance, ensuring that its efficacy remains undiminished. If performance aberrations or a discernible decline in predictive accuracy are detected, the model's recalibration procedure is invoked. This iterative mechanism guarantees that the model remains consistently at the forefront of accuracy and relevance, irrespective of evolving external parameters.

To achieve a comprehensive understanding of the flow of the proposed model, Algorithm 4.2 presents a systematic outline that covers the entirety of the process, starting from the extraction of

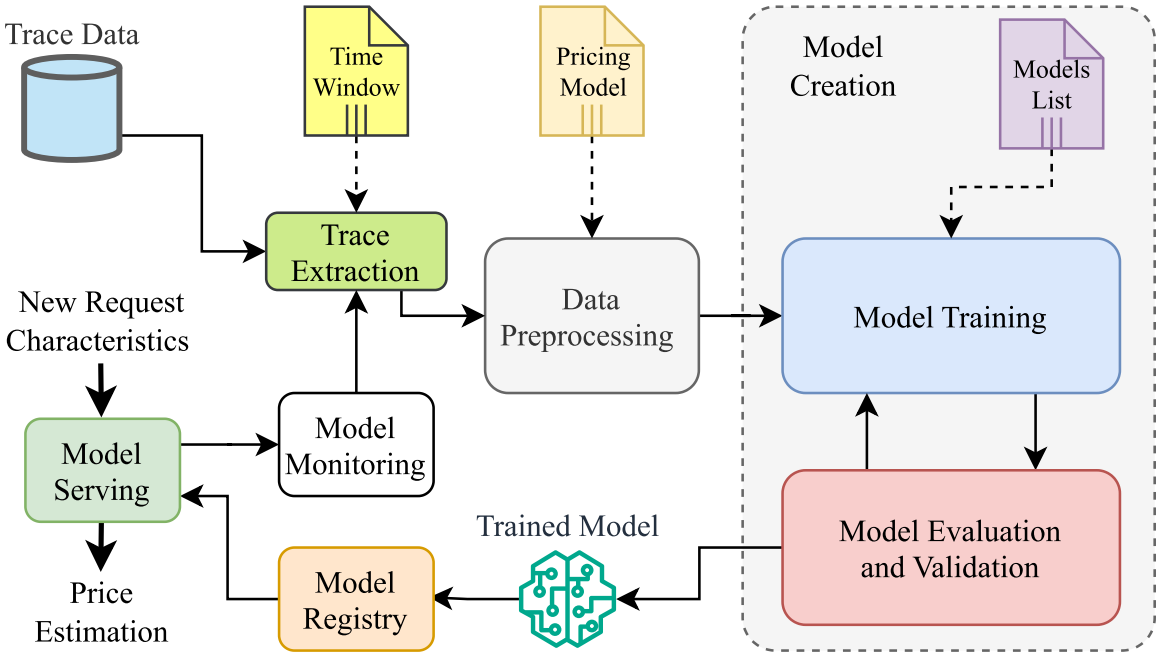

Figure 4.6: Schematic representation of the proposed model architecture.

Algorithm 4.2 Cloud Subscription Pricing Prediction Model

Require: Cloud subscription trace data S, window time W, candidate models M, threshold T, optimization metric m

Ensure: Predictive model for cloud subscription pricing

1: **Extract VMs Trace:**
2: VMs_data ← ExtractTrace(S, W)
3: **Data Preparation:**
4: VMs_lifetime ← ComputeLifetime(VMs_data)
5: TagUninterruptedVMs(VMs_data)
6: ComputeHourlyPriceForVMs(VMs_data)
7: ComputeTotalPrice(VMs_data, VMs_lifetime)
8: OneHotEncodeFeatures(VMs_data)
9: Aggregated_data ← GroupBySubscription(VMs_data)
10: Features, Target ← FeatureEngineering(Aggregated_data)
11: TransformedTarget ← TransformTarget(Target)
12: Train_data, Test_data ← SplitData(Features, TransformedTarget)
13: **Model Creation:**
14: Initialize ModelList as an empty list
15: **for** model in M **do**
16: FittedModel ← BuildAndEvaluate(model, Train_data, Test_data, m)
17: Append FittedModel to ModelList
18: **end for**
19: best_model ← SelectBestModel(ModelList, m)
20: **Model Serving:**
21: ModelRegistry(best_model)
22: **while** best_model is active **do**
23: MonitorPerformance(best_model, m)
24: **if** best_model performance drops below T **then**
25: Update W and S
26: **goto** Extract VMs Trace
27: **end if**
28: **end while**

the initial data and ending with the deployment of the model and its subsequent monitoring.

4.2.2 Data Preparation

As detailed in steps 2 to 10 of Algorithm 4.2, the data preparation phase plays a crucial role in determining the fundamental structure of the model. This stage meticulously transforms the raw data into a structured format, prepping them for ingestion by the subsequent model training and evaluation processes. The following elucidates the systematic steps undertaken during this phase:

(1) **Lifetime Computation:** The genesis of this stage involves computing the lifetime of each

VM. This process is achieved by discerning the difference between the creation and deletion timestamps. Such a computation is paramount as it underpins many subsequent calculations and helps to understand VM usage patterns.

(2) **Tagging Uninterrupted VMs:** After the lifetime computation, the data set undergoes a classification process where Uninterrupted VMs are identified and tagged. Recognizing these VMs is crucial, as they have unique usage patterns that can influence model prediction.

(3) **Pricing Calculation:** Each VM's hourly cost is computed next, considering various parameters, including its type and resource allocations. The aggregate cost of a VM over its entire lifetime is then determined by multiplying the hourly cost by the lifetime. This step translates resource utilization into monetary metrics, laying the foundation for cost predictions.

(4) **Categorical Feature Transformation:** The data set, at this point, often contains categorical variables such as the VM type, the core bucket, and the memory bucket. These categorical variables are subjected to a transformation procedure using the `OneHotEncoder` to improve their incorporation into ML models. This transformation converts the variables into a format more suitable for processing by a model.

(5) **Feature Aggregation and Engineering:** Data instances are grouped according to the subscription index for aggregation purposes. Various aggregation functions are utilized according to the characteristics of each feature. This procedure is critical in feature engineering because it manipulates the data set and determines the feature space and the target variable.

(6) **Target Transformation:** The dependent variable, denoting the subscription cost within a specified time frame (total cost of all allocated VMs), necessitates transformations to conform to the modeling assumptions and enhance the model's performance. This step ensures that the target variable is transformed to the optimal state for training using a predefined method.

(7) **Data Splitting:** The data set is divided into training and testing subsets as a final step in data preparation. This separation ensures a robust model evaluation with a training set that facilitates model learning and a testing set that serves as a benchmark for model performance.

4.2.3 VM Pricing Model

The pricing model presents an intricate yet systematic methodology to compute the hourly price of a VM. The model has several determinants: VM category, allocated core count, provisioned memory, and commitment lifetime duration. This segment delves into the mathematical underpinnings and foundational constructs of the model.

Central to this model are four crucial variables. The variable P_v is the base pricing corresponding to a VM of the category v. The multiplier C_c is adjusted for the core count specific to c. The multiplier M_m is aligned with the memory specification denoted by m. Finally, $L(d)$ is a function that measures the lifetime discount concerning a commitment that lasts for d days. With these variables in place, the hourly price, H, for a VM, can be concisely expressed as:

$$H(v, c, m, d) = P_v \times C_c \times M_m \times L(d) \tag{22}$$

In order to facilitate a thorough comprehension, Section 4.2.4.1 presents the precise values and elaborate illustrations for P_v, C_c, M_m, and $L(d)$. Thus, it exemplifies adaptability, allowing precise evaluations in various configurations.

4.2.4 Experimental Setup and Evaluation

The efficiency of the proposed model is evaluated using baseline regression models and various evaluation metrics. The evaluation results and additional details are described below.

4.2.4.1 Experimental Setup

Constants of the VM pricing model The proposed pricing model utilizes several constants to calculate the hourly price, denoted as $H(v, c, m, d)$, for using a VM. This price is influenced by various components, including the base hourly rate for the specific category of VM P_v, the multiplier associated with the number of cores C_c, the multiplier related to the memory specification M_m, and the discount multiplier for a lifetime commitment of d days $L(d)$.

Table 4.5 consolidates the base hourly prices, the core count multipliers, the memory multipliers, and the lifetime discounts, providing a comprehensive overview of the pricing structure for

Table 4.5: Pricing information for the proposed VM pricing model, including VM categories, CPU cores, memory, and lifetime discounts.

VM Category		Core Count		Memory (GB)		Lifetime (Days)		
Type (v)	**Base Pricing (P_v, per hour)**	**Count (c)**	**Multiplier (C_c)**	**Amount (m)**	**Multiplier (M_m)**	**Duration (d)**	**Multiplier ($L(d)$)**	**Discount**
General Purpose	\$0.02	2	1	2	1	<7	1.0	0%
Delay-insensitive	\$0.03	4	2	4	1.5	7 - 13	0.95	5%
Interactive	\$0.04	8	4	8	2	14 - 19	0.90	10%
-	-	24	6	32	2.5	≥20	0.85	15%
-	-	>24	8	64	3	-	-	-
-	-	-	-	>64	3.5	-	-	-

different VM categories, core counts, memory amounts, and commitment durations. It allows for the transparent and deterministic computation of the hourly pricing of a VM using equation 22, facilitating a clear understanding and application of the pricing model's components.

Workload and preparation setups Microsoft Azure offers a range of cloud computing services, including computing, storage, analytics, and networking. These services enable enterprises to implement and oversee cloud-based applications on a global scale [95]. Cloud users can submit multiple jobs to a regional data center using a single or multiple subscriptions. Each task is executed on a VM within a deployment, ensuring that the necessary resources are available for each task to operate optimally, considering its specific requirements.

Our experiments used a data set derived from the Azure Public Dataset v2 [96]. The duration of the trace is 30 consecutive days, including 6,687 Azure subscriptions with 2,695,548 VMs. The trace schema includes ordinal categories to show the CPU core and memory buckets of a VM and a nominal category representing the VM type. It also contains creation and deletion timestamps of VMs (used to calculate their lifetime), maximum and average CPU utilization, and the 95^{th} percentile of maximum CPU utilization. Figures 4.7 and 4.8 show, respectively, the distribution of lifespan and total price (lifetime in hours $\times$ price per hour) of VMs and the interrelations between different factors such as VMs' types, CPU cores, and memory buckets.

Algorithm 4.2, Steps 9 to 10 involve feature engineering related to subscriptions. This process computes the frequency of each categorical feature. Aggregates by subscription identifier, considering the average for numerical features and the sum of prices for all VMs as the target value. In Step 11, we examine the untransformed target to transformed one using the Yeo-Johnson and quantile methods based on *scikit-learn* implementation [9].

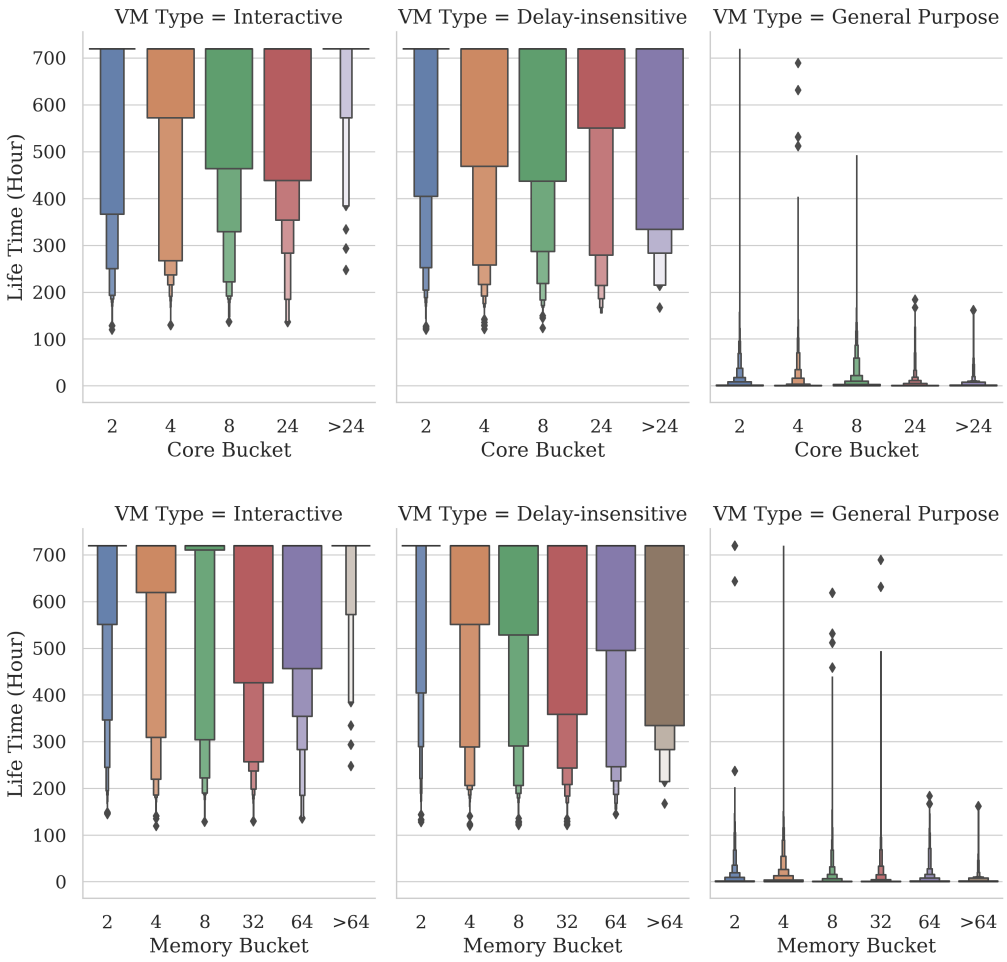

Figure 4.7: Boxen plot illustrating the distribution and interrelations among VMs' lifetime, type, CPU core count, and memory allocation.

Regression models and setups Our experiments compared the proposed model using 17 regression models of various types. All model configurations adhere to the default settings and implementations established by PyCaret [104], an open source ML library.

4.2.4.2 Evaluation Metrics

In order to ensure that the predictions are reliable and accurate, we employ a set of metrics that have been proven to evaluate performance effectively. The Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Coefficient of Determination (R^2), and Mean Absolute Percentage Error (MAPE), which are explained in details in Section 4.1.5.3. With

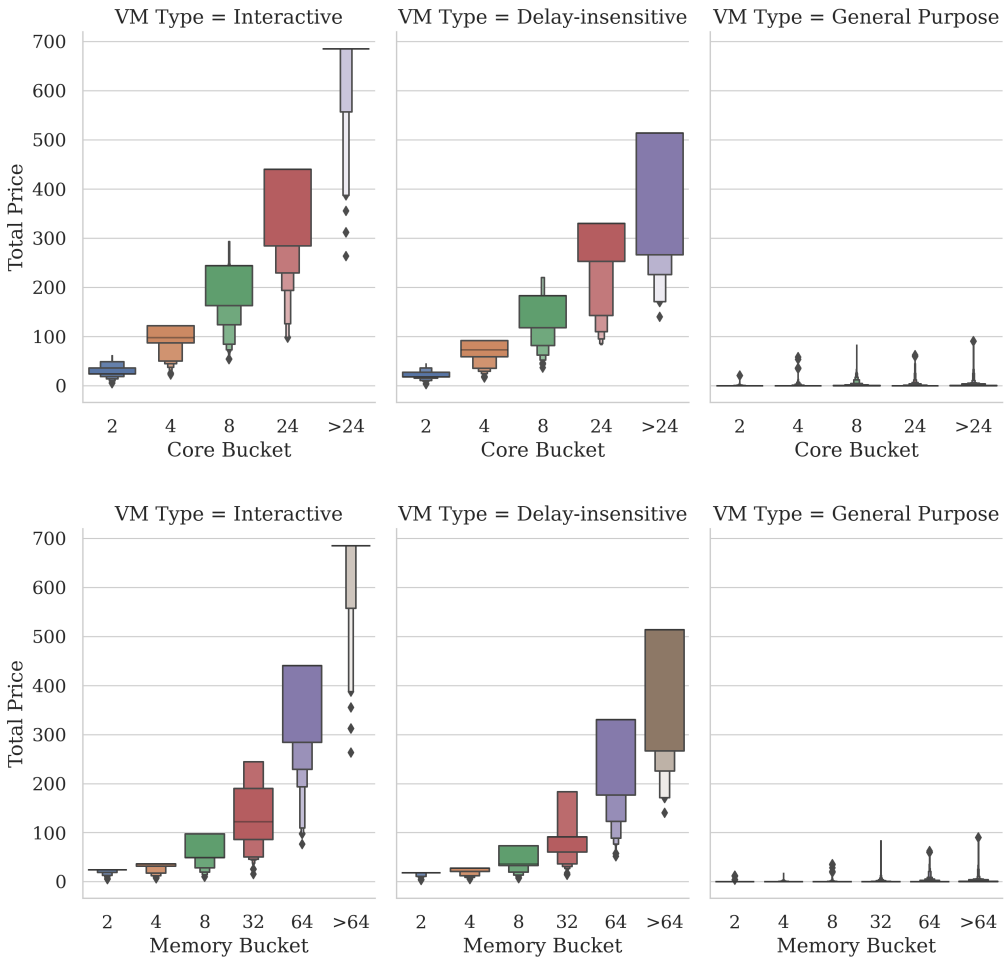

Figure 4.8: Boxen plot illustrating the distribution and interrelations among VMs' total price, type, CPU core count, and memory allocation.

these measurements, we can confidently assess the quality of the model's predictions.

4.2.4.3 Detailed Evaluation Results

Various regression models are meticulously evaluated to determine their predictive prowess in a workload trace derived from Microsoft Azure subscriptions. The employed models include CatBoost Regressor (CBR), Gradient Boosting Regressor (GBR), Extra Trees Regressor (ETR), Light Gradient Boosting Machine (LGBM), Random Forest Regressor (RFR), Extreme Gradient Boosting (XGB), Decision Tree Regressor (DTR), k-Nearest Neighbors (KNN), Elastic Net (ENet), Lasso Regression (Lasso), Linear Regression (LR), Lasso Least Angle Regression (LAR), Ridge

Table 4.6: Comparison of baseline regression models in the absence of target value transformation.

Model	MAE	MSE	RMSE	R^2	MAPE	TT (Sec)
CBR	364.024	6.364667e+06	2220.172	**0.893**	98.921	1.850
GBR	460.949	**5.893016e+06**	**2199.897**	0.891	212.761	0.297
ETR	**362.199**	6.808405e+06	2276.330	0.889	2.219	0.371
LGBM	434.902	6.575011e+06	2298.479	0.887	24.592	0.151
RFR	408.271	7.187883e+06	2376.625	0.880	2.491	0.963
XGB	408.565	7.085286e+06	2390.678	0.878	19.804	0.104
DTR	606.390	1.377838e+07	3480.050	0.716	**1.754**	0.025
KNR	712.620	1.618294e+07	3751.296	0.698	43.866	0.026
ENet	948.224	1.573679e+07	3715.407	0.693	1250.508	0.031
Lasso	948.190	1.573145e+07	3714.751	0.693	1249.030	0.035
LinR	947.209	1.587267e+07	3733.273	0.688	1243.444	0.726
LAR	951.135	1.590327e+07	3733.039	0.688	1190.015	0.018
Ridge	949.387	1.591703e+07	3735.930	0.687	1248.688	0.018
BRidge	949.220	1.593580e+07	3738.584	0.686	1272.094	0.019
AdaBoost	5743.391	3.926062e+07	6215.810	0.042	18909.765	0.107
DR	2501.602	5.010490e+07	6858.399	-0.003	6029.865	0.017
OMP	2446.427	5.425584e+07	6999.914	-0.024	5709.812	0.018

Note: Models are sorted using the R^2 score, and the TT values vary depending on the computational node used.

Regression (Ridge), Bayesian Ridge (BRidge), AdaBoost Regressor (AdaBoost), Dummy Regressor (DR), and Orthogonal Matching Pursuit (OMP). These models are selected based on their extensive utilization and the demonstrated efficacy in addressing diverse regression challenges. We thoroughly examine their predictive abilities concerning the subscription cost, considering different scenarios involving transforming target values, measured through a 10-fold cross-validation approach.

Scenario 1 (Absence of Target Value Transformation): In the given context, a thorough investigation is carried out to determine the effectiveness of a model when the target value is not transformed. The CBR approach surpasses its counterparts by achieving a low value of R^2. However, it is essential to acknowledge the significance of MSE and RMSE, as the dominance of GBR implies unparalleled reliability and precision in its predictive capabilities. In situations that require quick analytical responses, the LGBM algorithm stands out for its low Training Time (TT) among the top five R^2 score models, and it is essential for real-time applications. The ETR and DTR algorithms demonstrate exceptional performance in terms of MAE and MAPE metrics, respectively, showcasing their proficiency in reducing prediction inaccuracies.

Table 4.7: Comparison of baseline regression models with target value transformation using the Yeo-Johnson method.

Model	MAE	MSE	RMSE	R^2	MAPE	TT (Sec)
CBR	**0.258**	**0.220**	**0.468**	**0.976**	0.691	1.914
LGBM	0.277	0.248	0.497	0.973	0.681	0.134
ETR	0.269	0.277	0.525	0.969	**0.535**	0.322
XGB	0.299	0.283	0.531	0.969	0.760	0.136
RFR	0.278	0.298	0.544	0.967	0.636	0.703
GBR	0.352	0.320	0.565	0.965	1.523	0.295
DTR	0.413	0.645	0.802	0.929	0.474	0.027
AdaBoost	0.947	1.329	1.152	0.854	7.338	0.114
KNR	0.815	1.796	1.339	0.803	2.335	0.026
Lasso	1.911	6.796	2.601	0.255	15.154	0.042
LAR	1.911	6.793	2.601	0.255	15.152	0.024
ENet	1.909	6.862	2.612	0.247	14.965	0.039
BRidge	1.909	6.897	2.615	0.245	15.018	0.025
Ridge	1.908	6.919	2.619	0.242	14.950	0.021
LinR	1.909	7.101	2.645	0.223	14.940	0.021
DR	2.322	9.155	3.024	-0.002	22.303	0.017
OMP	2.318	9.906	3.123	-0.082	22.194	0.024

Note: Models are sorted using the R^2 score, and the TT values vary depending on the computational node used.

Scenario 2 (Incorporation of Yeo-Johnson Transformation): The Yeo-Johnson transformation has positively impacted the performance of CBR over other competing methods, underscoring the consistent and superior predictive accuracy and reliability. This scenario highlights the exceptional computational efficiency of LGBM among the top five R^2 score models, underscoring its suitability for applications that require rapid processing. Simultaneously, ETR exhibits a notable performance in terms of MAPE, demonstrating its effectiveness in reducing percentage errors, particularly in applications requiring high accuracy levels.

Scenario 3 (Incorporation of Quantile Transformation): By implementing the Quantile transformation method in this scenario, CBR maintains its prevailing position, showcasing unparalleled dependability and precision. The ETR algorithm is notable for its ability to achieve low MAPE values, indicating its proficiency in minimizing percentage errors. On the other hand, the LGBM algorithm demonstrates superior computational performance, making it particularly valuable for applications that require strict time limitations.

Conclusive Insights: The comprehensive examination of various scenarios reveals that using

Table 4.8: Comparison of baseline regression models with target value transformation using the Quantile method.

Model	MAE	MSE	RMSE	R^2	MAPE	TT (Sec)
CBR	**0.092**	**0.027**	**0.164**	**0.973**	0.412	2.011
LGBM	0.098	0.031	0.174	0.970	0.416	0.168
ETR	0.095	0.034	0.182	0.967	**0.328**	0.332
XGB	0.103	0.033	0.181	0.967	0.394	0.198
RFR	0.097	0.036	0.188	0.965	0.339	0.710
GBR	0.126	0.044	0.208	0.957	0.680	0.315
DTR	0.141	0.071	0.265	0.930	0.600	0.025
AdaBoost	0.313	0.161	0.401	0.842	1.854	0.106
KNR	0.269	0.193	0.439	0.810	1.248	0.024
LAR	0.670	0.769	0.875	0.245	1.693	0.018
Lasso	0.667	0.770	0.876	0.244	1.702	0.035
ENet	0.663	0.774	0.878	0.240	1.820	0.034
BRidge	0.663	0.792	0.886	0.224	1.920	0.020
Ridge	0.663	0.794	0.888	0.221	1.944	0.019
LinR	0.663	0.812	0.896	0.205	1.939	0.019
DR	0.801	1.020	1.009	-0.002	1.000	0.020
OMP	0.800	1.102	1.041	-0.075	1.017	0.018

Note: Models are sorted using the R^2 score, and the TT values vary depending on the computational node used.

the Yeo-Johnson transformation technique in Scenario 2 significantly improves the model's predictive capabilities. In this context, it is worth noting that using the CBR algorithm demonstrates exceptional accuracy and reliability. Table 4.9 provides strong evidence supporting the outstanding performance of CBR when applied with the Yeo-Johnson transformation. It highlights the consistent accuracy and reliability of CBR across different folds, making it a favorable option for a wide range of applications that require optimized and reliable results. Nevertheless, the uniform computational efficiency demonstrated by LGBM, including the balanced adaptability demonstrated by other models, provides a wide range of options that can be customized to meet different application requirements. The present synthesis of perspectives argues in favor of a methodical and comprehensive strategy for selecting models, considering accuracy, reliability, and computational efficiency.

Figure 4.9 illustrates the intricate dynamics of the learning curve, prediction error, and residuals for the paramount model in each delineated scenario, incorporating both the training and segregated hold-out data, albeit the deliberate omission of the k-fold cross-validation. In Scenarios 1, 2, and 3, the acquired R^2 values from training data are 1.000, 0.994, and 0.993, respectively, presenting a

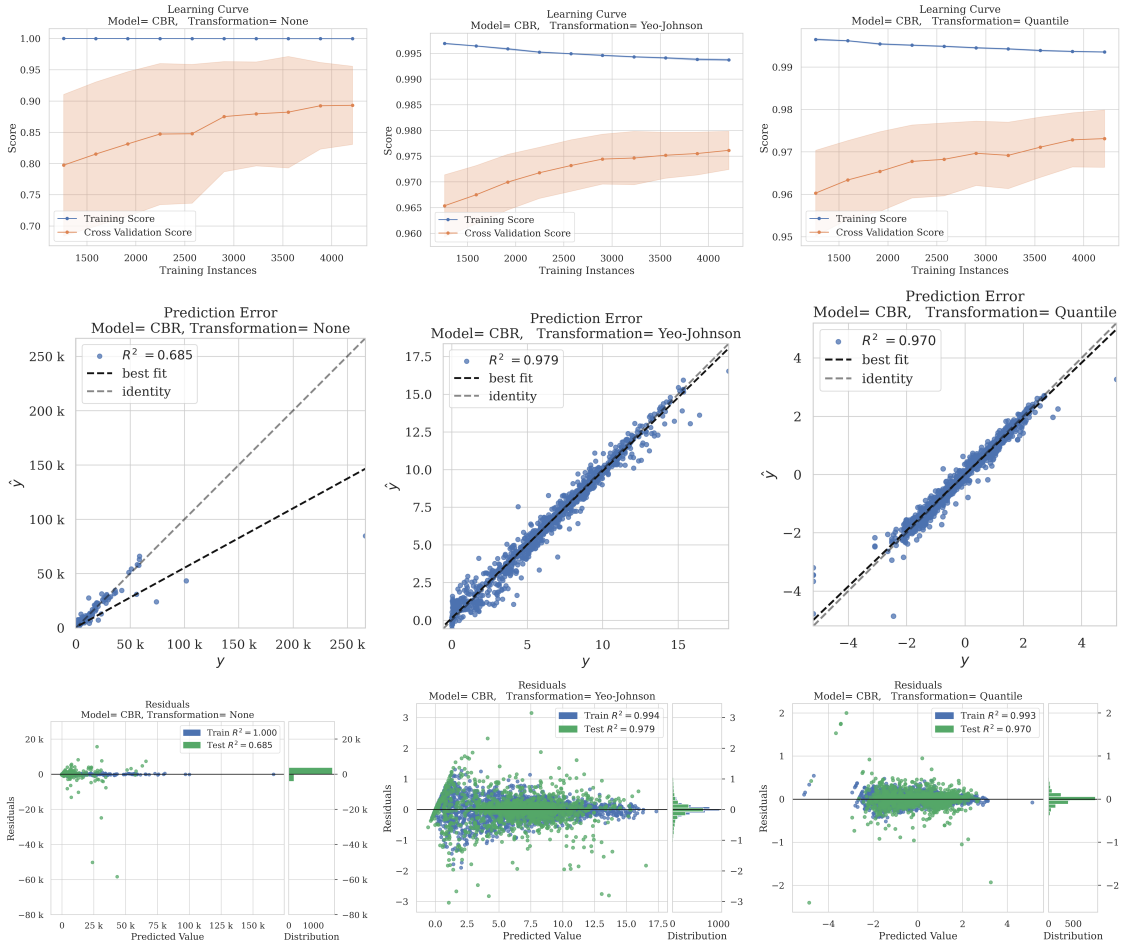

Figure 4.9: Comparative Analysis of Learning Curves, Prediction Errors, and Residuals Across Optimal Models in Different Target Transformation Scenarios.

stark contrast with the R^2 values from hold-out data, calculated as 0.685, 0.979, and 0.970. This conspicuous contrast in Scenario 1 unveils a substantial overfitting phenomenon, emphasized by the significant disparity in R^2 values between the meticulous training and the equally meticulous hold-out datasets. This overfitting phenomenon implies that the model, although performing with exemplary accuracy on the training set, fails to generalize effectively to unseen data (new data), indicative of a potential compromise in its predictive reliability. Interestingly, the overfitting is subtly alleviated by incorporating k-fold cross-validation, as seen in the learning curve, which can reduce model overfitting and foster a balanced, generalized performance.

Nonetheless, the strategic introduction of transformation techniques in Scenarios 2 and 3 effectively ameliorates the previously observed overfitting, substantiating the heightened efficacy of

Table 4.9: The 10-fold cross-validation scores for the CatBoost Regressor with target value transformation using the Yeo-Johnson method.

Fold	MAE	MSE	RMSE	R^2	MAPE
0	0.262	0.197	0.443	0.976	0.791
1	0.271	0.256	0.506	0.971	0.645
2	0.248	0.210	0.458	0.977	0.604
3	0.253	0.230	0.480	0.975	0.715
4	0.261	0.227	0.477	0.973	0.280
5	0.255	0.205	0.453	0.979	0.317
6	0.248	0.183	0.428	0.982	0.775
7	0.260	0.238	0.488	0.972	0.542
8	0.252	0.183	0.428	0.982	1.253
9	0.272	0.274	0.524	0.970	0.993
Mean	**0.258**	**0.220**	**0.468**	**0.976**	**0.691**
Std	**0.008**	**0.029**	**0.030**	**0.004**	**0.277**

CBR when synergistically aligned with the Yeo-Johnson transformation method. This correction is particularly pronounced in Scenario 2, where CBR achieves unparalleled accuracy and dependability, marking an R^2 of 0.979 on the hold-out data, thus reinforcing its supremacy amongst the plethora of evaluated models and transformation strategies.

The empirical evidence from these transformation scenarios unequivocally attests to the models' enhanced adaptability and predictive prowess, particularly CBR, under the transformative influence of evolved methodologies like the Yeo-Johnson transformation. This superior model performance under transformation accentuates the relevance and impact of employing transformation techniques. It underlines the importance of appropriately aligned methodologies to ensure optimal model generalization and predictive reliability across diverse and unseen datasets. These insights facilitate a deeper comprehension of the intrinsic model dynamics, the implications of overfitting, and the profound impact of transformation techniques, thereby guiding informed and enlightened model selection, deployment, and application in real-world scenarios.

4.2.5 Conclusion

In our previous research, we performed various studies using various prediction methods to forecast different aspects associated with resource planning and power management within cloud data centers [116]–[121]. However, this research has elucidated the complexities of forecasting financial

costs in cloud subscriptions and is published in [122]. Through employing various regression algorithms on real-world cloud workloads, this study has unveiled the potential for refined accuracy in predicting costs, enabling enhanced financial planning and strategic business decision-making. The proposed model serves as a cornerstone for addressing the challenges of predicting subscription costs, fostering transparency in cloud services, and benefiting service providers and consumers by mitigating risks related to improper pricing and resource allocation. Subsequent research will focus on integrating multilevel ensemble learning to enhance the model's adaptability across varied cloud service paradigms. The inclusion of diverse datasets is also planned to improve model robustness and precision in response to the complexities of cloud services. In conclusion, this study serves as a foundational step in predictive analytics for cloud computing, offering a path for innovations in optimizing cloud subscription pricing models.

4.3 Leveraging Imbalance and Ensemble Learning Methods for Improved Load Prediction in Cloud Computing Systems

Predicting load demand is critical to effective resource management in cloud computing. This process ensures optimal performance and efficiency by predicting the overload, underload, and normal load status of the physical machines. However, achieving accurate predictions remains challenging due to the highly dynamic and often non-linear workload patterns typical in cloud environments. Traditional methods, while helpful, have shown limitations in handling these complexities. Machine learning techniques, specifically imbalance and ensemble learning, have shown potential to improve prediction accuracy. Imbalance learning addresses the uneven distribution of load states, while ensemble learning combines multiple models to achieve better predictive performance. It is possible to create a more robust and accurate load prediction system by leveraging these two methods. This study explores the application of imbalance and ensemble learning to improve load prediction in cloud computing systems. Through an experimental study, we illustrate how these techniques outperform traditional methods, offering potential improvements to the performance and efficiency of cloud computing operations.

4.3.1 Host Load Characteristics

In cloud computing environments, the concept of host load is central to the effective management of resources. Three critical states define host load: overload, underload, and normal load. The following describes each state and its impact on cloud computing performance:

Overload State: When computing resources are insufficient to meet demand, an overload occurs, resulting in suboptimal system performance. This state can cause delays in task processing, system slowdowns, or failures, negatively impacting the user experience and harming the service provider's reputation. Furthermore, prolonged high loads can lead to hardware stress and early failure, which increases maintenance and replacement costs.

Underload State: On the other hand, when computing resources are not being used efficiently, it results in a state of underutilization. Although it may not immediately cause functional problems like overload, it still means that resources are not being allocated efficiently, leading to a waste of capacity. This state can negatively impact overall system efficiency and increase operating costs as hardware and energy resources are consumed without delivering adequate computational value.

Normal Load State: Maintaining a state of equilibrium is a fundamental requirement for any cloud computing infrastructure. This state ensures the resource supply aligns with the demand, facilitating optimal performance and efficiency. A stable load promotes seamless system operation, judicious energy consumption, and a gratifying user experience.

Predicting these load states accurately is essential for maintaining system performance and operational efficiency. Accurate load prediction allows for effective resource allocation and management, preventing overload and underload conditions and promoting sustained operation at or near the normal load state. However, due to the dynamic nature of cloud workloads, accurately predicting these states remains a significant challenge. The following sections discuss how imbalance and ensemble learning techniques can enhance load prediction accuracy in cloud computing environments. Accordingly, we propose a load prediction model based on a combination of imbalance and ensemble learning for cloud computing systems to improve the overall prediction accuracy.

4.3.2 Imbalance Learning for Load Prediction

Imbalanced learning deals with the issue of imbalanced data sets, where one class of data is greatly underrepresented compared to the others. This situation is common in cloud-based load prediction, where normal load states occur far more frequently than overload or underload states. Traditional ML algorithms often struggle with unbalanced data, tending to bias their predictions towards the majority class. Thus, it leads to poor predictive performance for the minority classes, which is problematic when these classes are of particular interest, as is the case with overload and underload states in cloud computing. The goal of imbalance learning is to fix any bias by either adjusting the class distribution in the training by resampling data or modifying the learning algorithm to focus more on the minority class at the algorithm level [123].

In resampling methods (our focus in this study), either the minority group is oversampled, the majority group is undersampled, or both are employed. Oversampling can be accomplished by duplicating minority class examples or creating synthetic examples, as in the Synthetic Minority Over-sampling Technique (SMOTE) [124]. In contrast, undersampling reduces the number of examples from the majority class. In the context of load prediction in cloud computing, imbalance learning can improve prediction accuracy by reducing the bias towards the normal load state. We can achieve more reliable predictions across all load states by ensuring the model pays sufficient attention to the overload and underload states. Thus, it can contribute to more effective resource management and higher system performance.

One possible example is implementing SMOTE with logistic regression for load prediction. It would involve creating synthetic examples of the minority class (overload and underload states) to balance the training data, then applying logistic regression to the balanced data to predict future load states. However, it is crucial to bear in mind that the choice of suitable methods for addressing imbalanced data sets is contingent on the specific characteristics of the data. Thus, achieving optimal results can involve a degree of trial and error to identify the most efficacious approach.

4.3.3 Ensemble Learning for Load Prediction

Ensemble learning is a highly effective type of ML where several models, also known as base learners, are trained to address the same problem and combined to achieve improved results. It is based on the principle that a group of weak learners can form a strong learner [125]. This approach enhances the model's generalization ability, prediction accuracy, robustness, and stability in the cloud-based load prediction context.

Load prediction can be complex due to cloud workloads' non-linear and dynamic nature. An ensemble of models, each trained on a different subset of the training data, can reduce the risk of overfitting (through bootstrap aggregating or bagging). For example, this method could involve creating an ensemble of decision trees where each tree is trained on a different bootstrap sample of the data. If the problem is that individual models are too simple and underfit the data (high bias), then boosting can be a helpful approach. Boosting involves training an ensemble of models sequentially, where each model tries to correct the mistakes of the previous ones. This method can increase the complexity of the overall model and reduce bias. An example might be using the AdaBoost algorithm for load prediction. Stacking involves training multiple models and using another ML model to combine their predictions. This method can capture each model's strengths and improve the predictions' robustness.

The voting classifier is another popular method for blending different ML algorithms. It works by training several models independently and then taking their predictions' mode for classification. The final prediction is the one that gets the most votes from all the classifiers. In a hard voting scheme, the final prediction is based on the most common output (majority vote) from all the classifiers. It does not consider the certainty of any individual model's prediction. In soft voting, on the other hand, the final prediction is based on the averaged probabilities calculated by each of the classifiers, giving more weight to highly confident class votes.

The appropriate choice of the ensemble learning method and base learners depends on the data's characteristics and the problem's specific requirements. For example, bagging might be more suitable if the data is highly noisy. If the base learners are too simple and underfit the data, boosting could help. If different models capture different patterns in the data, stacking might be beneficial.

However, the voting classifier can be particularly valuable in cloud-based load prediction, as it can leverage the strengths of different models to improve overall prediction accuracy. For example, one model might be good at predicting underload situations, while another might excel at predicting overload situations. Combining these models in a voting classifier can achieve more accurate and robust predictions across all load situations.

4.3.4 Imbalance and Ensemble Learning based-Load Prediction

Cloud-based load prediction can be challenging due to data imbalances. Minority classes like ”Under Load” and ”Over Load” can perform poorly, but imbalance and ensemble learning methods can create a more effective system. Imbalance learning addresses the issue by resampling data or adapting the model, while ensemble learning improves accuracy by combining multiple models. This study proposes a model for load prediction combining both techniques. Algorithm 4.3 outlines the process, which operates as follows:

Datasets Preparation: The first step is to use a set of imbalance learning techniques on the data. This step could involve oversampling the minority classes (’Under Load’ and ’Over Load’) or undersampling the majority class (’Normal Load’) to balance the data based on the given imbalance learning techniques list. The result is a variety of resampled data sets.

Creating Base Learners: After balancing the data using various imbalance learning methods, we train and fine-tune the given base learners in each data set. Then, the best-performing pipelines are chosen based on a given threshold of a selected performance metric. Each of these models’ pipelines can capture different patterns in the data, which can be used for a blending process.

Blending Process: This step combines the selected classifier pipelines using a voting classifier. The voting classifier utilizes the predictions from each pipeline and determines the final prediction by taking the majority vote, depending on the selected voting scheme (soft or hard). Then, the weights of the pipelines for the base classifiers are adjusted to optimize the score of the chosen performance metric. It guarantees that each base classifier plays a role in the ensemble process that leads to the greatest overall improvement in performance. By addressing the class imbalance issue and leveraging the strengths of multiple model pipelines, this approach provides a more sensitive and robust solution for load prediction.

Algorithm 4.3 Blended Imbalance and Ensemble Learning

Require: Datasets D, Performance metric m, Threshold t, Imbalance learning methods $B = \{b_1, b_2, ..., b_n\}$, Base classifier C, Voting scheme v
Ensure: Blended Classifier BC

```
for each imbalance learning method b_i in B do
    d_i ← Apply imbalance learning techniques b_i to D
    c_i ← Train C on d_i
    Tune hyperparameters of c_i to maximize m on d_i
    Compute performance metric m_i of c_i on d_i
    if m_i ≥ t then
        Add c_i to list of selected classifier pipeline S
    end if
end for
BC ← Voting Classifier(S, v)
Tune weights of BC to maximize performance metric m
    return BC
```

4.3.5 Experimental Setup and Evaluation

4.3.5.1 Performance Metrics

The model's ability to classify instances correctly in classification tasks can be evaluated using various performance metrics. According to [126], we used the most vital metrics to evaluate the proposed model based on the `scikit-learn` [9] implementation, which are outlined below.

Accuracy: It is a typical metric for evaluating classification model performance. It measures the percentage of correct predictions out of total predictions. It can be calculated as the number of correct predictions (True Positives (TP) plus True Negatives (TN)) divided by the total number of predictions (TP, TN, False Positives (FP), and False Negatives (FN)).

Precision: This metric measures how accurately a model predicts positive outcomes. Specifically, it calculates the ratio of true positive predictions to all positive predictions made by the model (TP/(TP + FP)). High precision suggests that the model can identify positive instances well but may not be equally proficient at identifying negative outcomes.

Recall: This metric determines how well the model can identify all positive cases in the data set. It is calculated by dividing the number of correct positive predictions made by the model by the total number of positive cases in the dataset (TP/(TP + FN). When a model has high recall, it means that it can recognize a lot of positive instances. However, this does not guarantee that it can

also avoid making false positive predictions.

F-measure: It combines precision and recall and is often used to provide a single overall score for the model's performance. There are several different ways to define an F-measure. The most commonly used is the F1 score, defined as the harmonic mean of precision and recall $F1 = 2 * (precision * recall)/(precision + recall)$. The F1 score is a measure of performance that considers both precision and recall. A higher score indicates better performance.

Area under the curve Regarding ML and statistical modeling, the AUC is a commonly used evaluation metric to measure the effectiveness of classification models. It is insensitive to the balance of the classes in the dataset and is not affected by the absolute values of the predicted probabilities. The idea behind it is the Receiver Operating Characteristic (ROC) curve, which is a graphical graph that illustrates the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) of a model at various classification thresholds. The AUC is calculated as the area under the ROC curve. It is a value between 0 and 1, with a higher value indicating better performance. An AUC of 0.5 corresponds to a model that is not better than random guessing, while an AUC of 1.0 corresponds to a perfect model that makes no mistakes.

Cohen's kappa score: The Cohen's kappa (kappa) score is a statistic used to measure the agreement between two annotators or raters who are classifying the instances. The kappa score is based on the proportion of agreement between the two annotators or raters, adjusted for the amount of agreement expected by chance alone. It can take on values from -1 to 1, with a value of 0 indicating no agreement beyond chance and a value of 1 indicating perfect agreement. The kappa score is calculated as follows:

$$\kappa = (p_o - p_e)/(1 - p_e) \tag{23}$$

where p_o is the proportion of items classified by annotators or raters, and p_e is the proportion of items expected to be classified the same way by chance alone. The kappa score is considered a more reliable measure of agreement than a simple percentage agreement because it considers the possibility of agreement occurring by chance.

Matthews correlation coefficient: The Matthews Correlation Coefficient (MCC) is employed in the field of ML to assess the efficacy of binary (two-class) classifications. The measure considers

both true and false positives and negatives and is commonly considered a balanced metric that can be applied even when the classes have significantly different sizes. The MCC is a correlation coefficient that ranges from -1 to +1. A coefficient of $+1$ indicates a perfect prediction, 0 signifies an average random prediction, and -1 denotes an opposite prediction. The statistic is alternatively referred to as the phi coefficient.

In the context of binary classification, tp, tn, fp, and fn represent the counts of correctly identified positive instances, correctly identified negative instances, incorrectly identified positive instances, and incorrectly identified negative instances, respectively. The MCC that can quantify the quality of binary classification predictions is calculated using the following formula:

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{24}$$

Multiclass evaluation: In multiclass classification, the goal is to classify instances into one of several classes. All of the above metrics can be used to evaluate the performance of a multiclass classification model. Accuracy, Precision, Recall, F1-score, AUC, and kappa scores can be calculated using either macro-averaging or micro-averaging. The macro-averaging involves calculating individual scores for each class and then computing their average. In contrast, micro-averaging calculates the scores globally by counting the total number of true positives, false positives, and false negatives. Both macro-averaging and micro-averaging have their advantages and disadvantages. Macro-averaging gives equal weight to each class, regardless of its size, while micro-averaging gives equal weight to each instance, regardless of its class. In general, macro-averaging is preferred when the classes are balanced, while micro-averaging is preferred when the classes are imbalanced.

For the MCC in the multiclass scenario, it can be precisely defined using a confusion matrix C for K classes. In order to provide a more concise explanation, we introduce the concept of intermediate variables as follows: $t_k = \sum_i^K C_{ik}$ (the number of times class k truly occurred), $p_k = \sum_i^K C_{ki}$ (the number of times class k was predicted), $c = \sum_k^K C_{kk}$ (the total number of samples correctly predicted), and $s = \sum_i^K \sum_j^K C_{ij}$ (the total number of samples). Then, accordingly, the multiclass MCC score is defined as:

$$MCC = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{(s^2 - \sum_k^K p_k^2) \times (s^2 - \sum_k^K t_k^2)}} \tag{25}$$

However, MCC will vary between -1 and $+1$ when there are more than two labels. The minimum value could potentially fall within the range of -1 and 0, depending on the quantity and arrangement of the truth labels. The maximum value is consistently $+1$.

4.3.5.2 Experimental setup

In our experiment setups, we utilized data from the CoMon project, monitoring over 1,000 VM on 800 hosts for two days running on CloudSim. We used a VM selection policy (minimum migration time) and a VM allocation policy (static threshold) proposed in [114]. We used a one-hour sliding window with 5 minutes as a utilization interval to capture the features alongside the date time indexes and total potential of CPU and main memory needs. Instances are classified as overloaded if the host's CPU utilization exceeds 95% during the upcoming interval, underloaded if below 10%, and normal load if it falls within these limits. The training dataset is split into three subsets using a time series cross-validator, each representing a different period. The model is then trained and evaluated three times, with different combinations of folds used as training and validation data. This method ensures that the model is tested on a diverse dataset, resulting in a more precise estimation of its performance.

For Algorithm 4.3 setup, we utilized a list of imbalance learning methods proposed in [127] to resample the original dataset. We used a logistic regression model as the base model classifier and a voting classifier as the blending classifier, provided in [9]. We also examine both hard and soft voting schemas of the voting classifier. The optimization of the models has been conducted using BayesSearchCV, provided in [9] as well, to maximize the F1 score as a performance metric. The selection of the imbalance learning models that perform best with the base classifier is based on the $F1$ score as an optimizing metric, with a threshold of 90%.

Table 4.10: Comparison of target class distribution and resampling time of different imbalance learning techniques

Fix Imbalance Method	Train Size	Normal-Load	Over-Load	Under-Load	RT
SMOTETomek	53359	37.38%	29.74%	32.88%	16.86
SMOTEENN	44539	35.04%	17.19%	47.77%	17.71
SVMSMOTE	45897	43.99%	17.99%	38.01%	36.05
BorderlineSMOTE	54795	37.68%	29.76%	32.56%	6.35
ADASYN	54681	37.75%	29.47%	32.78%	6.39
SMOTEN	54795	37.68%	29.76%	32.56%	124.79
SMOTE	54795	37.68%	29.76%	32.56%	5.89
RandomOverSampler	54795	37.68%	29.76%	32.56%	5.52
TomekLinks	24548	70.95%	2.77%	26.28%	6.75
RandomUnderSampler	2067	62.35%	9.48%	28.17%	5.09
OneSidedSelection	18503	89.03%	3.42%	7.55%	8.47
NeighbourhoodCleaningRule	22561	69.94%	2.96%	27.11%	7.8
NearMiss	2067	62.35%	9.48%	28.17%	4.84
InstanceHardnessThreshold	12642	63.01%	4.44%	32.55%	134.85
AllKNN	21204	69.63%	3.10%	27.27%	8.83
RepeatedEditedNearestNeighbours	19959	69.67%	3.24%	27.10%	11.25
EditedNearestNeighbours	20803	70.49%	3.14%	26.36%	6.07
CondensedNearestNeighbour	2777	71.90%	8.80%	19.30%	3221.16
NA	25774	70.82%	2.67%	26.51%	6.41

4.3.5.3 Evaluation Results

Table 4.10 compares various techniques for imbalance methods regarding target class distribution and Resampling Time (RT). Table 4.11 shows the performance of a logistic regression classifier under various imbalance methods, sorted based on the $F1$ score, including the Training Time (TT). We have selected the top nine models for the blending process based on their $F1$ scores, which are higher than the given threshold. Table 4.12 shows the cross-validation scores for a soft-voting classifier. It can be observed that the classifier has performed consistently across the three folds with a minimal standard deviation in the performance metrics with mean outperformed the other methods. Figure 4.10 shows AUC and the trade-off between the precision and the recall of the soft-voting classifier, with 0.96 of the weighted average precision.

4.3.6 Conclusion

This study explored the fusion of ensemble and imbalance learning to improve load prediction in cloud computing systems. The proposed model uses a combination of both techniques to

Table 4.11: Comparison of logistic classifier utilizing various imbalance methods, sorted by F1 score, including the train time

Fix Imbalance Method	AUC	Accuracy	F1	Kappa	MCC	Precision	Recall	TT
AllKNN	0.979	0.9454	0.9344	0.8683	0.8701	0.9337	0.9454	1.25
NeighbourhoodCleaningRule	0.9799	0.9457	0.9342	0.8694	0.8711	0.9338	0.9457	1.01
NA	0.9805	0.9458	0.9336	0.8696	0.8714	0.9349	0.9458	0.88
EditedNearestNeighbours	0.9795	0.9444	0.9332	0.8662	0.8679	0.9334	0.9444	1.2
TomekLinks	0.9802	0.945	0.9331	0.8672	0.8693	0.9324	0.945	1.14
RepeatedEditedNearestNeighbours	0.9797	0.9439	0.9331	0.8653	0.8668	0.9323	0.9439	0.96
SMOTEN	0.9618	0.9158	0.9186	0.8094	0.8118	0.9225	0.9158	2.07
InstanceHardnessThreshold	0.9359	0.9279	0.9156	0.8309	0.834	0.9058	0.9279	0.15
SVMSMOTE	0.9803	0.9076	0.9141	0.7991	0.8053	0.9268	0.9076	4.14
SMOTEENN	0.9789	0.8801	0.8983	0.7528	0.7665	0.9292	0.8801	3.72
SMOTE	0.9769	0.8729	0.8965	0.7415	0.7578	0.9346	0.8729	3.19
RandomUnderSampler	0.9735	0.8742	0.8959	0.7444	0.7602	0.9314	0.8742	0.22
SMOTETomek	0.9767	0.8694	0.8947	0.7354	0.7526	0.9354	0.8694	4.33
BorderlineSMOTE	0.9774	0.8679	0.8908	0.735	0.7535	0.9306	0.8679	2.85
ADASYN	0.9779	0.8623	0.8869	0.7241	0.7436	0.9294	0.8623	5.34
RandomOverSampler	0.977	0.8116	0.8549	0.6497	0.6898	0.9347	0.8116	5.6
CondensedNearestNeighbour	0.9097	0.7957	0.8311	0.5978	0.6169	0.8885	0.7957	0.26
OneSidedSelection	0.9072	0.7524	0.786	0.5166	0.5492	0.8742	0.7524	1.47
NearMiss	0.9682	0.6603	0.6807	0.4329	0.5084	0.8428	0.6603	0.3

Table 4.12: Soft-based voting classifier cross-validation scores

Fold	Accuracy	AUC	Recall	Precision	F1	Kappa	MCC
0	0.9432	0.9805	0.9432	0.9348	0.9376	0.8656	0.8661
1	0.9465	0.9833	0.9465	0.9425	0.9442	0.8753	0.8754
2	0.9407	0.9781	0.9407	0.9309	0.9342	0.8591	0.8597
Mean	0.9435	0.9806	0.9435	0.9361	0.9387	0.8667	0.8671
Std	0.0024	0.0021	0.0024	0.0048	0.0042	0.0067	0.0065

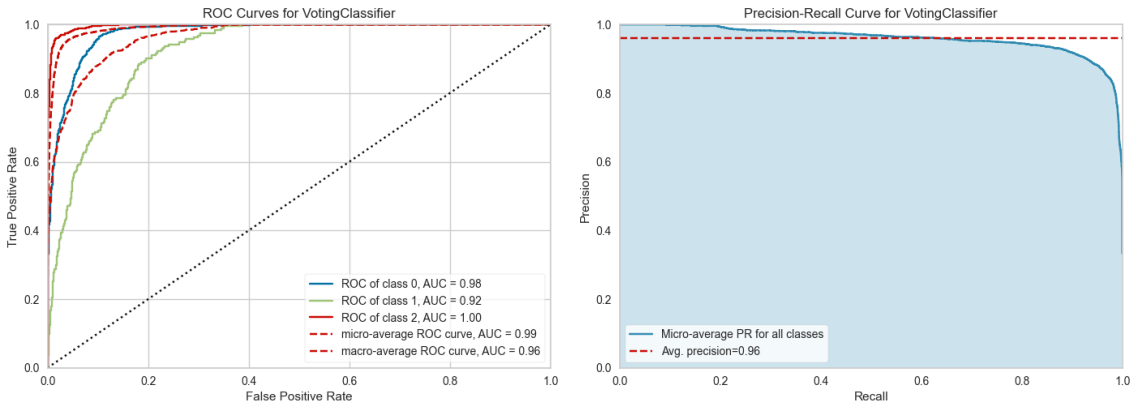

Figure 4.10: AUC and precision/recall plots for soft-based voting classifier

improve sensitivity and reliability in load prediction. Although our approach showed promising results, several potential directions could further enhance the prediction of host loads. Incorporating more diverse base classifiers into the ensemble could lead to more robust predictions. Instead of a simple voting classifier, more advanced ensemble methods could be explored to better leverage the strengths of individual models. This research is published in [128] and opened new avenues for predicting host load in cloud computing. By further refinement, we anticipate even greater improvements in prediction accuracy, thus optimizing resource allocation and enhancing the overall efficiency of cloud computing systems.

Chapter 5

Innovative Strategies in Multi-Output Predictive Modeling for Serverless Computing

This chapter introduces a comprehensive suite of innovations to improve the predictability of invocation within serverless architectures. By employing multi-output regression models, we perform a multilevel analysis of invocation patterns across user, application, and function levels, revealing insights into granular workload behaviors. We investigate the impact of windowing techniques and the reduction in dimensionality on model performance considering PCA, offering a nuanced understanding of the complexities of data and the computational implications. The proposed comparative framework meticulously evaluates the performance considering various windowing configurations, utilizing the Azure Functions dataset for real-world applicability.

5.1 Introduction

Serverless computing, also known as Function-as-a-Service (FaaS), has become a cornerstone in the evolution of cloud computing [70]. FaaS is a cloud computing model in which providers dynamically manage resource allocation, allowing users to run event-driven applications without the complexity of managing the underlying infrastructure [129]. This model is recognized for its

efficient utilization and cost-effectiveness, offering a flexible solution for many applications [79].

The adoption of native cloud technologies, such as microservices and containers, has dramatically increased the popularity of serverless computing as an architectural choice and programming model. Detailed discussions on this topic can be found in [130] and [75], which also explore the context of serverless architecture orchestration. However, as a relatively recent innovation in the cloud computing landscape, serverless computing is still navigating its early stages of exploration and development. This growth phase aims to address critical challenges, particularly by improving the predictability and efficiency of function invocations, which are receiving more research attention [84]. Therefore, the burgeoning field of serverless computing represents fertile ground for research, especially in optimizing and refining these core aspects [131].

Serverless computing, although increasingly popular, faces various challenges, specifically in effectively handling the dynamic and unpredictable nature of function invocation [70]. The inherent unpredictability of the system adds complexity to resource management and performance optimization, which are particularly crucial in the context of High-Performance Computing (HPC) [132]. HPC entails performing computations at a higher performance level than general-purpose computing. This model requires a more resilient and efficient management system in serverless environments [133]. The practice of *warming up* functions, which involves prepping functions in advance for anticipated execution, is an approach to tackle these intricacies [81].

5.2 Methodological Framework

5.2.1 Analyzing Function Invocation at Different Levels

This section explains how to analyze function invocation in serverless computing at different levels. The methodology covers three primary levels: the user, the application, and the function. Each level of analysis is critical for understanding the various aspects of function invocation and their impact on the serverless environment.

5.2.1.1 Data Preparation for Analysis Perspectives

The trace data serves as an essential component of software systems that enable analysts to comprehend the system's operations and pinpoint areas for improvement. In cloud data centers, the trace data refers to the set of records of events and activities that occur within the system, and it is systematically collected through a monitoring agent. This agent is connected to a workflow engine, which manages the sequence of tasks in a process, and a FaaS system, which allows the execution of functions in response to specific events.

In this study, the monitoring agent systematically records the rate at which invocations of functions occur over time at regular intervals. It keeps track of each function's invocation frequency within a designated time period. The raw data set of the time series $\mathcal{D}$ is represented by tuples consisting of $(h_o, h_a, h_f, T, c_1, c_2, \ldots, c_N)$ for each record. In this context, h_o represents the identifier for the user, h_a represents the identifier for the application, and h_f represents the identifier for the function. The value of T represents the type of trigger, which indicates the event that caused the function to be called. The variable c_i represents the number of times the function was called during the i^{th} time interval from the N observations within a designated period. The raw data set $\mathcal{D}$ is manipulated to facilitate analysis from various perspectives: User, application, and function. The preparation involves aggregating the invocation counts based on different grouping criteria:

(1) **User Perspective:** At the user level, we examine how users interact with the serverless platform, focusing on the frequency and patterns of function invocations. Thus, $\mathcal{D}$ is aggregated by user and trigger type (excluding h_a and h_f) to focus on usage patterns at the user level. The resulting data set D_{h_o} is defined as:

$$D_{h_o} = \sum_{h_o,T} c_i \quad \forall i \in \text{invocation counts intervals, grouped by } h_o \text{ and } T \tag{26}$$

The summation is applied to all N function invocation counts for each unique combination of user and trigger type, with h_o subsequently excluded to emphasize trigger-based aggregation.

(2) **Application Perspective:** Application-level analysis focuses on how serverless applications,

as a whole, invoke functions. Aggregation is performed by application and trigger type (excluded h_o and h_f) to understand application-specific invocation patterns. The resulting data set D_{h_a} is given by:

$$D_{h_a} = \sum_{h_a,T} c_i \quad \forall i \in \text{invocation counts intervals, grouped by } h_a \text{ and } T \tag{27}$$

Similarly to D_{h_o}, the summation is applied to all invocation counts of functions N, focusing on combinations of application and trigger type, with h_a subsequently excluded.

(3) **Function Perspective:** The function level focuses on the invocation characteristics of individual functions. Function-level performance analysis involves examining the data without considering the specific user or application context. The resulting data set D_{h_f} focuses solely on trigger type and invocation counts:

$$D_{h_f} = \{T, c_1, c_2, \ldots, c_N\} \quad \text{excluding } h_o, h_a, \text{ and } h_f \tag{28}$$

This structured data preparation process transforms the raw time series data set $\mathcal{D}$ into subsets of time series data sets that facilitate the targeted analysis of user behavior, application performance, and function utilization, providing a comprehensive view of the system's operational dynamics.

However, the produced subsets time series data sets (D_{h_o}, D_{h_a}, and D_{h_f}) go through window sliding operation (detailed below) that transform the time series data into patterns that suit for the multi-output regression models. This windowing operation enables a more accurate and insightful analysis of the system's operational dynamics by transforming the time-series data into patterns suitable for our intended analysis.

5.2.1.2 Window Sliding Operation for Invocation Time Series Data

The windowing process is a fundamental technique used in the analysis of time series data, particularly useful in ML and signal processing. Consider a discrete time series $\{x_t\}_{t=1}^{N}$ where x_t represents the value of the series at time t and N is the total number of observations. The objective

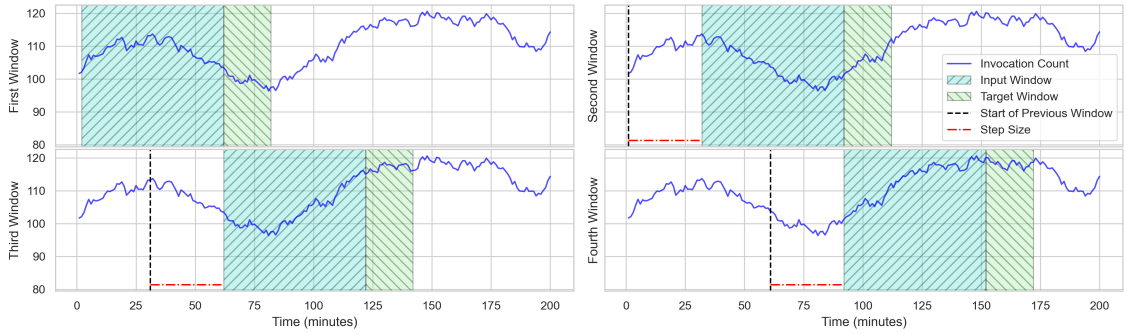

Figure 5.1: Sequential illustration of the windowing process applied to synthetic time series data. The figure displays four consecutive windows, with turquoise patterned areas representing input windows (z_1, z_2, z_3, z_4) and green patterned areas indicating corresponding target windows (y_1, y_2, y_3, y_4). The red dashed lines mark the initiation of new windows, underscoring the step size between them.

of windowing is to transform this series into a set of overlapping subsequences that can be used to predict subsequent values or detect patterns.

A window, often called a frame, is defined by two parameters: the window size W_z and the step size S. The window size W_z determines the number of data points included in each subsequence, while the step size S dictates the displacement between the start of consecutive windows. The target window size W_y specifies the subsequence length we aim to predict.

Given the window size W_z, the step size S, and the target window size W_y, the i-th input window is represented as $\{x_i, x_{i+1}, ..., x_{i+W_z-1}\}$, and the corresponding target window is given by $\{x_{i+W_z}, x_{i+W_z+1}, ..., x_{i+W_z+W_y-1}\}$. The process iterates over the entire series, starting at $i = 1$ and increasing with the size of the step S to $i + W_z + W_y - 1 \leq N$.

Figure 5.1 illustrates the windowing process applied to synthetic data. It facilitates an intuitive grasp of how the windowing process prepares time series data for further analytical tasks. Four consecutive windows are depicted to demonstrate the time series segmentation into input and target windows. Vertical dashed lines delineate the boundaries of each window, and the patterned areas signify the input and target data points within the time series. The progression of the windowing process is visually indicated by the red dashed line, which represents the step size.

The window sliding operation, described in Algorithm 5.1, is crucial in converting unprocessed

Algorithm 5.1 Window Sliding Operation

Require: ts (invocation time series data), W_z (window size), S (step size), W_y (target window size)

Initialize $features\_list$, $targets\_list$
if $step\_size < 1$ **then**
    $step\_size \leftarrow 1$
end if
$R \leftarrow$ Compute range of iteration: [ts_{start} to $ts_{end} - (W_z + W_y)$] with steps of S
for $start\_slice$ in R **do**
    $end\_slice \leftarrow start\_slice + W_z$
    $target\_slice \leftarrow end\_slice + W_y$
    Extract feature and target windows from ts using $start\_slice$, end_slice, and $target\_slice$
    Add the corresponding Trigger type T into feature windows
    Append feature and target windows to $features\_list$ and $targets\_list$
end for
Concatenate $features\_list$ and $targets\_list$ into $X\_Pattern$ and $y\_Pattern$
Remove the duplicate patterns, considering combined $X\_Pattern$ and $y\_Pattern$

Ensure: Return $X\_Pattern$ and $y\_Pattern$

time series data into organized patterns appropriate for multi-output regression models. This technique is precious in serverless computing, where understanding the temporal dynamics of function invocation at different levels is vital for enhancing performance and managing resources.

The algorithm begins by initializing the lists to hold the features and targets for each window. It then iterates over the time series data, segmenting it into overlapping windows of size W_z with a step size of S and a corresponding target window of size W_y. For each iteration, it extracts the relevant feature and target windows, adding the corresponding trigger type T to enhance the data context. The final result is a pair of data sets $X_{Pattern}$ and $y_{Pattern}$, respectively, consisting of features and target patterns, with duplicates removed to ensure uniqueness. However, the application of the window sliding operation to time series data is fundamental for comprehensively analyzing serverless systems for several reasons:

- **Extraction of Temporal Features:** This technique is highly effective in extracting time-related characteristics from data, accurately capturing the patterns and timing of function invocations. Extracting this information is crucial for understanding the frequency, order, and timing of function calls, which are essential to predict the behavior of invocation at the related level and optimize performance accordingly.

- **Pattern Identification and Prediction:** This method is instrumental in identifying recurrent and unique patterns in function invocation, providing a basis for detecting regularities and predicting future system states. These patterns are invaluable for forecasting workloads, identifying potential bottlenecks, and preventing system failures or anomalies. By actively removing redundant patterns, this approach ensures the uniqueness and significance of the identified patterns. This refinement step improves the predictive accuracy and efficiency of the system by focusing on distinct informative patterns that contribute to a more nuanced understanding of system behavior and potential future states. It helps to maintain a streamlined pattern data, reducing noise, and improving the speed and quality of the analytics.

- **Structural Transformation for Analytical Modeling:** By restructuring raw time series data into a sequence of overlapping windows, the operation converts variable-length sequences into a suitable consistent format for analysis. This structured approach is not only beneficial for ML algorithms but is also essential for multi-output models, which rely on a fixed-size input to forecast multiple points. The resultant structured data is thus more amenable to a wide variety of analytical models, enhancing the accuracy and insight of the analysis.

- **Generic Modeling:** The sliding window technique facilitates the creation of a single generic model applicable to multiple time series, diverging from the traditional approach in which each series requires its own model. This method significantly improves scalability and flexibility. This process promotes more efficient resource utilization and enables faster adaptation to new data or changing conditions in serverless environments. By reducing the need for individual model tuning and maintenance, it streamlines the analytical process, making it more robust and adaptable to various types of time series data.

In essence, the window sliding technique transforms the subsets of time series data sets (D_{h_o}, D_{h_a}, and D_{h_f}) into patterns aligned with the analytical objectives of multi-output regression models intended in this study. This transformation is essential to dissecting and understanding the intricate operational dynamics of serverless systems, enabling stakeholders to make informed decisions based on comprehensive temporal analysis and predictions.

Studies such as [134], [135], and [136] have emphasized the importance of windowing techniques in time series analysis, particularly in the context of cloud computing, serverless, and predictive modeling. Adapting this technique to serverless computing allows a deeper understanding of the dynamics of function invocation, which is essential to optimize performance, manage resources effectively, and reduce operational costs.

In conclusion, the window sliding operation detailed in Algorithm 5.1 is a foundational step in the preparation of the time series datasets for in-depth analysis. Systematically segmenting the data into meaningful subsequences facilitates a range of analytical tasks and enhances our understanding of the intricate dynamics of serverless systems at user, application, and function levels, fostering a more precise prediction mechanism.

5.2.1.3 Invocation Pattern Preparation for Multi-Output Regression Models

For the practical application of multi-output regression models in serverless computing, it is imperative to prepare the invocation patterns that align with the predictive modeling requirements. The window sliding operation detailed in Algorithm 5.1 produces two key data sets: $X_{Pattern}$, representing the features of the invocation patterns, and $y_{Pattern}$, representing the corresponding target. These data sets are further refined to fit the structure and demands of multi-output regression models. The preparation process is described methodically in Algorithm 5.2, which explains how to transform the invocation patterns into a suitable format for predictive modeling. This process involves a series of steps that include encoding, feature selection, and dimensionality reduction.

Initially, the invocation pattern datasets $X_{Pattern}$ and $y_{Pattern}$, derived from the window sliding operation, are split into training and testing sets based on the specified test size and random state. This division is crucial for assessing the model's performance and ensuring that it can generalize well to new data. After splitting, OneHotEncoder is initialized to convert the categorical trigger type T feature in $X_{Pattern}$ into a one-hot numeric array, effectively transforming categorical data into a numerical format appropriate for regression analysis. The OneHotEncoder is fitted in X_{train} and then used to transform both X_{train} and X_{test}, replacing the trigger column with encoded features. This process ensures that categorical variables are appropriately handled in the model.

Subsequently, a VarianceThreshold is initialized and fitted X_{train} to remove all features with

Algorithm 5.2 Prepare Data for Multi-Output Regression Models

Require: $X_{Pattern}$, $y_{Pattern}$, $apply\_pca$, $variance\_threshold$, $test\_size$, $random\_state$, $objects\_path$

1: Split $X_{Pattern}$ and $y_{Pattern}$ into X_{train}, X_{test}, y_{train}, y_{test} random subsets based on $test\_size$ and $random\_state$
2: Initialize OneHotEncoder ▷ To encode categorical features as a one-hot numeric array
3: Fit OneHotEncoder on X_train and transform X_train and X_test ▷ Update X_train and X_test by replacing 'Trigger' with encoded features
4: Save OneHotEncoder object into $objects\_path$ ▷ To be used for the Hold-Out Data
5: Initialize VarianceThreshold with (threshold = 0.0) ▷ To keep all features with non-zero variance
6: Save VarianceThreshold object into $objects\_path$ ▷ To be used for the Hold-Out Data
7: Fit VarianceThreshold on X_train and transform X_train and X_test
8: **if** $apply\_pca$ is True **then**
9: Initialize $PCA\_Transformer$ with $variance\_threshold$ and $random\_state$ ▷ To keep only the first PCA components where the cumulative sum of explained variance ratio exceeds $variance\_threshold$
10: Fit $PCA\_Transformer$ on X_train and transform X_train and X_test
11: Save $PCA\_Transformer$ object into $objects\_path$ ▷ To be used for the Hold-Out Data
12: **end if**

Ensure: Return transformed X_{train}, X_{test}, y_{train}, y_{test}

Algorithm 5.3 Prepare Hold-Out Data for Multi-Output Regression Models

Require: $X_{holdout}$, $y_{holdout}$, $objects\_path$, $apply\_pca$

1: Load $OneHotEncoder$ from $objects\_path$
2: Transform $X_{holdout}$ using the fitted $OneHotEncoder$ ▷ Encode 'Trigger' feature as done in training
3: Load $VarianceThreshold$ from $objects\_path$
4: Transform $X_{holdout}$ using the fitted $VarianceThreshold$ ▷ Apply Variance Threshold as done in training
5: **if** $apply\_pca$ is True **then**
6: Load $PCA\_Transformer$ from $objects\_path$
7: Transform $X_{holdout}$ using the fitted $PCA\_Transformer$ ▷ Apply PCA transformation as done in training
8: **end if**

Ensure: Return transformed $X_{holdout}$, $y_{holdout}$

zero variance, effectively streamlining the input data by eliminating redundant or noninformative variables. This step is crucial for enhancing the model's performance by focusing on relevant features. The fitted VarianceThreshold is then used to transform both X_{train} and X_{test}, ensuring consistency in the feature space between the training and testing data sets. If PCA is to be applied, indicated by $apply\_pca$ being True, a $PCA\_Transformer$ is initialized with the specified variance

threshold and random state. This transformer is fitted to X_{train} to identify the principal components that cumulatively explain a proportion of variance that exceeds the given variance threshold. Then both X_{train} and X_{test} are transformed using this fitted $PCA\_Transformer$, reducing the dimensionality of the data while attempting to preserve as much of the original variance as possible.

Ensuring consistency and reliability in model predictions requires applying the same transformation process to hold-out data as was used during training. As described in Algorithm 5.3, the fitted $OneHotEncoder$, $VarianceThreshold$, and $PCA_Transformer$ from the training phase are used to systematically transform the hold-out data $X_{holdout}$. This method includes encoding categorical features, removing features with zero variance, and applying dimensionality reduction, each using only the transform method of the respective object. Such a consistent application of transformations ensures that the inputs to the model maintain uniform structure and scale across training and hold-out data sets, leading to reliable predictions. This rigorous adherence to a standardized transformation process is pivotal for maintaining the integrity and efficacy of multi-output regression models, especially in the dynamic contexts of serverless computing environments.

In conclusion, Algorithms 5.2 and 5.3 systematically process the data to ensure that multi-output regression models receive input that are encoded, selected, and dimensionally reduced in a manner that maximizes their predictive performance. This preparation is fundamental to leveraging the full potential of multi-output regression models in serverless computing, providing insights into function invocation patterns and aiding in optimizing and effectively managing serverless architectures.

5.2.2 Adaptive Optimization Framework for Serverless Time Series Analysis

In the domain of serverless computing, understanding and optimizing the performance of multi-output regression models is crucial for effective resource management and service quality. This section outlines a comprehensive methodology for training, evaluating, and optimizing these models on time series data, particularly focusing on serverless function invocation patterns. The methodology relies on two key algorithms: Algorithm 5.4 for training and evaluating models and Algorithm 5.5 for optimizing the window sliding parameters in time series patterns.

The algorithm 5.4 describes the process of training and evaluating a multi-output regression model on time series patterns. Initially, the data is prepared using Algorithm 5.2, which ensures that

Algorithm 5.4 Train and Evaluate Multi-Output Regression Model for Time Series Patterns

Require: $X_{Pattern}$, $y_{Pattern}$, $model$, $metric$, $data\_prepare\_param$, $models\_path$

1: Prepare $X_{Pattern}$, $y_{Pattern}$ based on $data\_prepare\_param$ and get X_{train}, X_{test}, y_{train}, y_{test}
▷ Using Algorithm 5.2
2: Train $model$ on X_{train} and y_{train}
3: Save $model$ into $models\_path$
4: Predict with $model$ on X_{test} to obtain y_{pred}
5: Replace all negative values in y_{pred} with zeros
6: Convert y_{pred} to integer
7: Calculate $score$ using metric(y_{test}, y_{pred})

Ensure: Return the calculated $score$

$X_{Pattern}$ and $y_{Pattern}$ are appropriately transformed for the modeling process. The model is then trained on the resultant X_{train} and y_{train} datasets. After training, the model is saved to a specified path for future inference or comparison. Predictions are made on the testing data set X_{test}, and post-processing steps are applied to ensure that the predictions are in the correct format (e.g., non-negative and integer values). Finally, the performance of the model is evaluated using a specified metric, which compares the predicted values y_{pred} with the actual values y_{test}.

The performance of time series models can often be significantly affected by the choice of parameters for window sliding. The algorithm 5.5 outlines an approach to optimize these parameters for a given time series data set ts. It iteratively explores combinations of window sizes W_z, step sizes S, and target window sizes W_y within the specified $window\_param\_ranges$. For each combination of parameters, the algorithm applies window sliding to the time series data to generate $X_{Pattern}$ and $y_{Pattern}$ using Algorithm 5.1. Each generated pair of feature and target data sets is then evaluated using Algorithm 5.4, which trains a model on the data and calculates a score representing the model's performance. These scores are collected for all combinations of parameters, and the best performing combination is determined based on whether the goal is to minimize or maximize the given performance metric.

Subsequently, the most effective arrangement and model derived from Algorithm 5.5 are implemented on data that has not been previously observed. This deployment entails the execution of the model on each of the subsets generated from time series data sets (D_{h_o}, D_{h_a}, and D_{h_f}), which were previously defined to represent different levels of analysis from the user, application, and function perspectives, respectively. Deploying the model is crucial to evaluate its ability to perform well and

Algorithm 5.5 Optimize Window Sliding Parameters for Time Series Patterns

Require: ts, $window\_param\_ranges$, $model$, $metric$, $data\_prepare\_param$, $optimize$

Initialize $scores$ as an empty list
for each window size W_z in $window\_param\_ranges['window\_size']$ **do**
    for each step size S in $window\_param\_ranges['step\_size']$ **do**
        for each target window size W_y in $window\_param\_ranges['target\_window\_size']$ **do**
            Apply window sliding to ts with W_z, S, W_y and gets $X_{Pattern}$ and $y_{Pattern}$ ▷ Using Algorithm 5.1
            Evaluate $model$ on $X_{Pattern}$, $y_{Pattern}$, & $data\_prepare\_param$ and get $score$ ▷ Using Algorithm 5.4
            Append $(score, W_z, S, W_y, model, data\_prepare\_param)$ to $scores$
        end for
    end for
end for
if $optimize$ is 'minimize' **then**
    Sort $scores$ in ascending order
else
    Sort $scores$ in descending order
end if
return $scores[:1]$ ▷ Return the best performing configurations

be effective in a live serverless environment, replicating real-life scenarios where the model will generate predictions on new data.

In a dynamic serverless environment, the performance of deployed models must be continuously monitored to ensure sustained effectiveness. As function invocation patterns and workload characteristics may change over time, it is critical to closely track the model's performance metrics. If a performance drop is detected, indicating a possible shift in the underlying data patterns or their distribution, the entire process from data preparation to model optimization will be re-applied using the most recent invocation data. This re-application ensures that the model stays updated and aligned with the latest trends and changes in the serverless architecture, maintaining its accuracy and reliability. Continuous monitoring and periodic re-optimization embody a proactive approach to maintaining model performance, thereby ensuring that predictive capabilities are always tuned to the highest standard of efficiency and effectiveness.

To summarize, the comparative analysis methodology offers a strong framework for constructing multi-output regression models within the realm of serverless computing at different levels of

analysis perspectives. The methodology improves the understanding and effectiveness of the models in time series data by methodically preparing the data, training the models, and optimizing the window sliding parameters. The deployment of the optimized model on unseen data provides additional assurance that the model is theoretically sound and practically effective. This approach plays a crucial role in advancing serverless computing by facilitating a more efficient allocation of resources, enhancing service quality, and improving predictive capabilities.

5.3 Experimental Results and Analysis

5.3.1 Azure Functions Workload

The study presented in [137] performs an in-depth analysis of the production Azure Functions workload. It provides critical insights into the characteristics and invocation frequencies of real-world functions, illuminating the operational demands placed on cloud service providers. An important observation from the study is the notably short duration of functions within the FaaS workload, particularly compared to other cloud workloads. For example, data from Azure VM workload [96] indicates that while 63% of all VM allocations last longer than 15 minutes, less than 8% of VMs persist for 5 minutes or less. This stark contrast highlights the unique challenges of FaaS environments, which impose strict requirements on providers for rapid resource allocation and scaling. Such quick turnaround times are essential to meet the dynamic and fleeting nature of serverless function executions, necessitating efficient and agile infrastructure management.

The study also underscores the importance of comprehensively characterizing the production FaaS workload. It encompasses an array of parameters, including fundamental function types, trigger mechanisms, frequency of invocations, and the corresponding resource requirements. Addressing the scarcity of publicly available data on real-world FaaS workloads, the study emphasizes the need for detailed information on the cumulative demand faced by cloud providers. It also explores the challenges of managing cold starts and proposes a predictive policy employing time series analysis techniques such as ARIMA modeling. This policy aims to forecast subsequent invocations and optimize resource allocation accordingly.

In the context of this study, the Azure Functions data set version 2019 presented in [137] is

used, which provides granular data on the invocation counts of functions, the execution durations, and the metrics of memory usage. A specific focus is placed on analyzing the function invocation counts at different levels to discern usage patterns and operational characteristics. The data set comprises multiple files, each representing function invocations over 24 hours. Collectively, these files provide detailed insights into function usage and operational characteristics over a long period. The analysis in this study predominantly uses the first 12 days of this dataset. We employ the first day of the dataset for training and testing the model, ensuring a robust foundation for model development. The subsequent 11 days are utilized as hold-out data, providing a comprehensive and extended assessment of the model's predictive accuracy and generalization capabilities on unseen function invocation patterns. The schema of the data set includes the following fields for each day.

- **HashOwner**: Identifier for the owner of the application (represented as h_o in our model).
- **HashApp**: Identifier for the application (represented as h_a in our model).
- **HashFunction**: Identifier for the function within the application (represented as h_f in our model).
- **Trigger**: Trigger that initiates the function, classified into various types (represented as T in our model).
- **[1 .. 1440]**: Columns representing the number of invocations per minute for 24 hours (represented as $(c_1, c_2, \ldots, c_N)$ in our model).

In particular, all identifiers are unique and hashed using *HMAC-SHA256* with secret salts to maintain consistency between data, enabling association between owners, applications, and functions. The *Trigger* field categorizes the function's initiation mechanism into seven distinct groups, including HTTP, timer, event, queue, storage, orchestration, and others. The invocation fields provide a minute-by-minute account of function executions, offering a detailed view of usage patterns and demands on the Azure Functions infrastructure. This data set is invaluable for researchers who want to optimize performance or better understand the behaviors of cloud functions.

5.3.2 Azure Functions Workload Analysis with Multi-Output Regression Models

The comprehensive study of Azure Function workload provides a pivotal foundation for applying advanced analytical methods. Notably, multi-output regression models emerge as a powerful tool to analyze function invocation patterns across various levels of granularity in serverless computing. Our proposed methodology employs these models to dissect and understand the intricate dynamics of function invocation at the user, application, and function levels, each providing unique insights into the serverless environment.

The function invocation counts derived from the Azure Functions dataset have significant value beyond their numerical representation. These figures are a critical indicator of operational efficiency and performance within a serverless architecture. As such, they are critical metrics that provide insight into the operational rhythm of the system. Therefore, it is essential to carefully monitor and analyze these metrics to optimize the performance of the serverless architecture for efficient and effective operations. By applying multi-output regression models to these counts, we can predict future invocation patterns and resource needs more accurately. This predictive capability is vital for proactive resource allocation and efficient management of serverless architectures, as it helps anticipate and mitigate potential bottlenecks and performance degradation.

The Azure Functions data set is rich and enables a comprehensive analysis at multiple levels: user level (denoted as D_{h_o} in our model), application level (denoted as D_{h_a} in our model), and function level (denoted as D_{h_f} in our model). These levels correspond to different time series obtained using Equations 26, 27, and 28, respectively. Every level of analysis is indispensable to develop a comprehensive understanding of the serverless workload. Through the utilization of multi-output regression models, we conduct a detailed analysis that captures the temporal dependencies and patterns that are inherent in invocation data. This approach enables cloud providers and system architects to make informed decisions about resource allocation, system scaling, and performance optimization. Our proposed methodology provides a structured approach to dissecting the serverless workload, emphasizing the adaptive and predictive aspects of modern cloud services.

5.3.3 Evaluation Metrics

Evaluation metrics are crucial for quantifying the efficacy of predictive models. In this study, we used MAE, MSE, and R^2 outlined in Section 4.1.5.3. For multi-output regression problems, the score of each metric is computed for each output separately, and an average over all outputs is used to obtain a single performance measure. These metrics provide a multifaceted view of model performance. They are critical in our evaluation analysis to ensure that our models are not only accurate on average MAE but also sensitive to the magnitude of errors by examining MSE and capable of explaining the variance in target output patterns by estimating R^2. Scikit-learn offers robust implementations for these metrics, providing standardized and efficient criteria for our model evaluations and facilitating the reproducibility and comparability of our results.

5.3.4 Experimental Setup

In the context of our adaptive optimization framework, the selection of appropriate window parameters is critical for an effective time-series analysis of serverless function invocations. The `window_param_ranges` dictionary defines the range of values over which the window sliding parameters will be optimized. Specifically, it includes:

- **Window size (W_z)**: This parameter determines the length of each window or segment of the time series data that will be considered for analysis. In our study, the window size is varied among *[30, 60, 90, 120]* minutes, allowing us to understand how the choice of window size impacts the model's ability to capture relevant patterns in function invocations.

- **Step size (S)**: This parameter specifies the step or displacement between consecutive windows. A smaller step size means higher overlap between consecutive windows and more fine-grained analysis, while a larger step size reduces the computational load at the expense of granularity. Our step sizes are set at *[1, 15, 30]* minutes, ensuring a range from high overlap to moderate overlap.

- **Target window size (W_y)**: This refers to the size of the window to predict future values. It is essential to determine how far ahead the model should forecast. We consider target window

sizes of *[30, 60]* minutes to explore short-term predictions within the scope of the behavior of serverless functions at each level.

By iterating over these ranges, the optimization Algorithm 5.5 systematically explores various configurations of window sliding, assessing which combination best captures the temporal dynamics of serverless function invocations while balancing the trade-off between computational efficiency and predictive accuracy.

In our adaptive optimization framework, the data preparation phase transforms the raw time series data into a suitable format for multi-output regression modeling. The `data_prepare_param` dictionary encapsulates the key parameters that guide this phase, ensuring that each data set is appropriately processed before being fed into the model. Specifically, it includes:

- $apply\_pca$: The parameter is set to either [False, True], indicating whether PCA should be applied or not. In this study, we examine both cases in which applying PCA can mitigate the curse of dimensionality and improve the performance of the model by focusing on the components that account for the most variance in the data.

- **variance_threshold**: This parameter, set at 0.99, defines the amount of variance that needs to be captured by the selected components in PCA. A high threshold like 0.99 means that the model will attempt to retain the components that together explain 99% of the variance, ensuring that most of the original information is preserved while still benefiting from reduced feature space.

- **test_size**: This parameter dictates the proportion of the data set to be reserved for testing. Here, a value of 0.2 indicates that 20% of the data will be used as a test set while the remaining 80% will constitute the training set. This split is crucial for evaluating the model's performance and ensuring that it generalizes well to unseen data.

By adjusting these parameters, researchers and practitioners can fine-tune the data preparation process to suit the specific needs and constraints of their analytical tasks. The `data_prepare_param` dictionary therefore plays a vital role in setting up the data for subsequent modeling and analysis, directly impacting the efficiency and effectiveness of the predictive models.

Table 5.1: Detailed Parameters for Inherently Multi-Output Regression Models

Index	Model	Parameters
LR	Linear Regression	copy_X: True, fit_intercept: True, positive: False
KNN	K-Neighbors Regressor	algorithm: auto, leaf_size: 30, metric: minkowski, metric_params: None, n_neighbors: 5, p: 2, weights: uniform
DT	Decision Tree Regressor	ccp_alpha: 0.0, criterion: squared_error, max_depth: None, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, min_samples_leaf: 1, min_samples_split: 2, min_weight_fraction_leaf: 0.0, random_state: 123, splitter: best
RF	Random Forest Regressor	bootstrap: True, ccp_alpha: 0.0, criterion: squared_error, max_depth: None, max_features: 1.0, max_leaf_nodes: None, max_samples: None, min_impurity_decrease: 0.0, min_samples_leaf: 1, min_samples_split: 2, min_weight_fraction_leaf: 0.0, n_estimators: 100, oob_score: False, random_state: 123, warm_start: False
ET	Extra Trees Regressor	bootstrap: False, ccp_alpha: 0.0, criterion: squared_error, max_depth: None, max_features: 1.0, max_leaf_nodes: None, max_samples: None, min_impurity_decrease: 0.0, min_samples_leaf: 1, min_samples_split: 2, min_weight_fraction_leaf: 0.0, n_estimators: 100, oob_score: False, random_state: 123, warm_start: False

In our quest to identify the most effective windowing and data preparation configurations for serverless time series, we have used the LR model from the Scikit-learn library as a foundational modeling technique. LR model is inherently capable of multi-output regression, making it particularly suited for our scenario where predicting multiple future invocation counts is necessary.

We have opted for the default configuration of the LR model provided by Scikit-learn. By leveraging the simplicity and effectiveness of this model, we aim to dissect and understand the relationship between the windowing parameters and the predictive accuracy of our models. This approach allows us to systematically explore and optimize the window size, step size, target window size, and data preparation configurations, ensuring that our final setup is well tuned to provide accurate and timely predictions for function invocations in serverless computing environments.

Building on the best configuration identified in the preceding step, this study employs, in addition to LR, a suite of regression models for in-depth analysis, each initialized with its default settings as specified by Scikit-learn (version 1.2.2). These models are inherently designed for multi-output regression tasks, encompassing KNN, DTR, RFR, and ETR regressors. Each model has been carefully selected for its suitability to handle complex multidimensional output without further implementation, providing a robust analytical framework. The specific parameters and their respective

default values for each model are meticulously documented in Table 5.1, providing a comprehensive overview of the models' configurations used in this study. This methodology guarantees a consistent and adaptable analytical setting, facilitating accurate and reliable regression analysis.

5.3.5 Comparative Analysis of Window Sliding Parameters

This section delves into a comprehensive comparative analysis of window sliding parameters at different levels of function invocation within serverless computing environments with and without the application of dimensionality reduction. By dissecting the user, application, and function levels, we understand how predictive modeling can be optimized across various aspects of serverless architecture. The use of LR in this comparison allows a straightforward interpretation of the relationship between input features and predicted results, making it a suitable choice to understand the impact of window sliding parameters on model performance. In addition, it reduces the computation costs associated with more complex models.

The analysis is presented methodically through Tables 5.2, 5.3, and 5.4, each illustrating the implications of employing PCA on the predictive precision. Furthermore, columns P_c and P_c' denote the original and reduced counts of data points, respectively, illustrating the volume of data being processed and the resulting reduction by removing the redantant patterns according to the given window sliding parameters.

5.3.5.1 User-Level Analysis

The user level analysis, shown in Table 5.2, is critical for understanding how individual behavior impacts the invocation of serverless functions and thereby influences the overall system's performance. By examining the window sliding parameters in invocation time series data at the User level, we aim to optimize predictive models that can accurately forecast user interactions with serverless functions. The analysis is classified based on the application of PCA, a technique crucial in managing the dimensionality of the data, thus affecting the complexity and efficiency of the predictive model.

Without PCA (Table 5.2a): The model without PCA provides a baseline utilizing the full spectrum of data dimensions. This approach offers a detailed view of user interactions but at a

higher computational cost. The highest performing configuration, as indicated by the lowest MAE value of **22.4001**, is observed with a window size (W_z) of **120**, a step size (S) of **1**, and a target window size (W_y) of **60**. Furthermore, the volume of patterns that are processed (P_C) of **19101628** and the resulting reduction (P_c') of **7307047** with a reduction percentage of around **61.74%** due to the large number of redundant patterns. This outcome indicates that a finer temporal resolution and a more extensive historical context lead to more accurate predictions at the user level. However, as we move down the table, increasing S and decreasing W_z or W_y generally correspond to an increase in MAE, suggesting a loss of critical information, which is detrimental to model performance.

With PCA (Table 5.2b): The PCA-applied model aims to reduce the computational load by simplifying the data feature set while attempting to retain the most significant variance within the data. The optimal configuration under PCA shows a MAE of **21.7611**, achieved with a window size (W_z) of **120**, a step size (S) of **30**, and a target window size (W_y) of **30**. Also, the volume of patterns that are processed (P_C) of **666512** and the resulting reduction (P_c') of **256037** with a reduction percentage of around **61.74%**. Interestingly, this configuration, despite the reduced dimensionality and volume of the pattern, outperforms the best non-PCA model in terms of MAE, underscoring the effectiveness of PCA in enhancing model performance by focusing on the most influential data features and the characteristic uniqueness of the patterns. However, the trade-off between dimensionality reduction and loss of detail is evident across various configurations, emphasizing the need for a balanced approach in model construction and feature selection.

Implications of Findings: The detailed comparative analysis at the user level reveals several important implications. First, the choice of window sliding parameters has a profound impact on the predictive accuracy of the models, with larger window sizes generally providing more context for prediction but requiring careful consideration of the step and target sizes to maintain model performance. Second, applying PCA can significantly improve computational efficiency and, in some configurations, even improve predictive precision by eliminating redundant information. However, its application must be judicious, ensuring that the reduction in dimensionality does not overlook critical behavioral patterns essential for accurate prediction. Lastly, the sorted results based on MAE provide a clear hierarchy of the performance of the model, guiding the selection of the appropriate configurations based on the specific needs for accuracy and computational resources.

Table 5.2: Comparative Analysis of Window Sliding Parameters in Invocation Time Series Data at User Level

(a) PCA = False

MAE	W_z	S	W_y	P_c	P'_c
22.4001	120	1	60	19101628	7307047
22.6566	120	1	30	19556068	7043090
23.4433	120	30	30	666512	256037
23.6819	120	15	30	1317876	499528
23.7114	120	15	60	1287580	516113
24.1967	90	30	30	681660	244458
24.2066	90	1	60	19556068	7043090
24.6357	90	1	30	20010508	6669682
24.8298	90	15	60	1317876	499528
24.8914	90	30	60	666512	256037
24.9393	90	15	30	1348172	476038
25.4298	120	30	60	651364	264341
26.1004	60	1	60	20010508	6669682
26.1917	60	30	60	681660	244458
26.4095	60	15	60	1348172	476038
26.5360	60	30	30	696808	227051
26.9603	60	15	30	1378468	440733
27.7661	60	1	30	20464948	6119737
29.2018	30	15	60	1378468	440733
29.3698	30	30	60	696808	227051
30.0785	30	1	60	20464948	6119737
30.9767	30	15	30	1408764	390280
32.3219	30	30	30	711956	201494
34.1617	30	1	30	20919388	5384934

(b) PCA = True

MAE	W_z	S	W_y	P_c	P'_c
21.7611	120	30	30	666512	256037
22.5365	120	15	60	1287580	516113
22.7309	120	15	30	1317876	499528
22.9182	90	30	30	681660	244458
23.2235	120	1	60	19101628	7307047
23.4465	90	30	60	666512	256037
23.5943	120	30	60	651364	264341
23.6823	120	1	30	19556068	7043090
23.7199	90	15	60	1317876	499528
24.1475	90	15	30	1348172	476038
24.9023	60	30	60	681660	244458
25.4551	90	1	60	19556068	7043090
25.5474	60	30	30	696808	227051
25.6770	60	15	60	1348172	476038
26.0935	90	1	30	20010508	6669682
26.5803	60	15	30	1378468	440733
27.7015	60	1	60	20010508	6669682
29.0088	30	30	60	696808	227051
29.3909	30	15	60	1378468	440733
29.7438	60	1	30	20464948	6119737
31.1230	30	1	60	20464948	6119737
31.3388	30	30	30	711956	201494
31.4411	30	15	30	1408764	390280
35.1279	30	1	30	20919388	5384934

Future Directions: Moving forward, further research could explore the dynamic adaptation of window sliding parameters and PCA components based on user behavior patterns, considering segmentation as a prior process like the work presented in [101], [107]. This process could lead to more flexible and accurate predictive models capable of adapting to the evolving nature of user interactions in serverless environments. Furthermore, investigating hybrid models that integrate multilevel learning strategies and cost estimation techniques, such as those introduced in [115] and [122] respectively, could provide novel avenues for enhancing the predictability and cost-effectiveness of serverless computing systems at the user level.

5.3.5.2 Application-Level Analysis

Table 5.3 shows a comprehensive comparative analysis of window sliding parameters at the application level. The focus shifts to how applications comprising multiple functions interact within the serverless environment, both with and without the application of dimensionality reduction. Accordingly, we can dissect and understand the collective behavior of functions as they interact within specific applications. This level is crucial for understanding and optimizing resource allocation, scalability, and overall performance of serverless systems.

Without PCA (Table 5.3a): The application-level analysis without PCA serves as a complete representation of the complexity of all the data. The configuration with the highest performance in terms of predictive precision is marked by a MAE of **18.4511**, achieved with a window size (W_z) of **120**, a step size (S) of **1**, and a target window size (W_y) of **60**. This configuration reflects that higher data degrades computational efficiency. As the table progresses, the variations in W_z, S, and W_y show the corresponding changes in the MAE, indicating the sensitivity of the model performance to these parameters. In the best-performing configuration, the count of original data points (P_c) is **27700387**, while the reduced data points (P_c') is **9274343**. This reduction shows that the windowing approach is also highly efficient in managing the volume of data at the application level, reducing the size by around **66.51%**. This decrease in data points leads to a more streamlined prediction.

With PCA (Table 5.3b): When PCA is applied, the analysis progresses toward understanding how dimensionality reduction affects predictive modeling at the application level. The most effective configuration results in a reduced MAE of **16.9978**, indicating an improvement in precision. This configuration is achieved with a window size (W_z) of **120**, a step size (S) of **30**, and a target window size (W_y) of **30**. The reduction in data volume is also notable, with PCA further reducing the complexity of the data set while maintaining quality. The results indicate that PCA effectively balances the need for computational efficiency with the requirement of accurate predictions at the application level. In the most optimal setup, the number of original data points (P_c) is **944581**, whereas the reduced data points (P_c') amount to **338099**. This decrease demonstrates that the windowing technique is also extremely effective in handling the amount of data at the application level when PCA is applied, resulting in a reduction of approximately **64.20%** in size.

Table 5.3: Comparative Analysis of Window Sliding Parameters in Invocation Time Series Data at Application Level

(a) PCA = False

MAE	W_z	S	W_y	P_c	P_c'
18.4511	120	1	60	27700387	9274343
18.6181	120	15	30	1911129	635095
18.6752	120	30	60	944581	338099
18.7141	120	1	30	28359397	8892705
19.3863	120	15	60	1867195	659296
19.5960	120	30	30	966548	325965
19.6540	90	15	60	1911129	635095
20.0009	90	1	60	28359397	8892705
20.5056	90	1	30	29018407	8365574
20.6778	90	15	30	1955063	601542
20.7204	90	30	30	988515	309345
20.7650	90	30	60	966548	325965
21.2986	60	15	60	1955063	601542
21.6550	60	30	30	1010482	284899
21.7345	60	1	60	29018407	8365574
22.0113	60	30	60	988515	309345
22.7276	60	15	30	1998997	552107
23.0547	30	30	60	1010482	284899
23.0890	60	1	30	29677417	7608574
24.4084	30	15	60	1998997	552107
24.7596	30	1	60	29677417	7608574
26.9295	30	30	30	1032449	250048
27.1122	30	15	30	2042931	483687
28.6774	30	1	30	30336427	6624745

(b) PCA = True

MAE	W_z	S	W_y	P_c	P_c'
16.9978	120	30	60	944581	338099
17.2195	120	15	30	1911129	635095
17.8888	120	30	30	966548	325965
18.1780	120	15	60	1867195	659296
18.7491	90	15	60	1911129	635095
19.0811	120	1	60	27700387	9274343
19.3624	90	30	60	966548	325965
19.4455	90	30	30	988515	309345
19.5242	120	1	30	28359397	8892705
19.7444	90	15	30	1955063	601542
20.6265	60	30	30	1010482	284899
20.6860	60	15	60	1955063	601542
20.7000	60	30	60	988515	309345
20.7488	90	1	60	28359397	8892705
21.4177	90	1	30	29018407	8365574
22.2539	60	15	30	1998997	552107
22.5325	30	30	60	1010482	284899
22.8589	60	1	60	29018407	8365574
24.5272	60	1	30	29677417	7608574
24.5407	30	15	60	1998997	552107
25.5793	30	1	60	29677417	7608574
26.4422	30	30	30	1032449	250048
27.3422	30	15	30	2042931	483687
29.3699	30	1	30	30336427	6624745

Implications of Findings: The comparative analysis at the application level underscores several key points. First, the choice of window sliding parameters significantly influences predictive performance, with larger windows providing more historical context but also requiring more computational resources. Second, PCA's role in reducing dimensionality is beneficial in terms of computational efficiency and can lead to improved accuracy in certain configurations. However, the delicate balance between data reduction and the preservation of essential information is crucial. The results offer a guide for selecting the right configuration based on specific needs with respect to accuracy and computational constraints.

Future Directions: Future research might explore adaptive window sliding and dimensionality reduction techniques, perhaps exploring how these parameters can be dynamically adjusted based

on changing patterns of application usage. Similarly to the user level, further investigation of hybrid and advanced modeling techniques could also provide more nuanced insights into predictive accuracy and computational efficiency at the application level. Additionally, incorporating real-time learning and update mechanisms into these models could improve their adaptability and responsiveness to evolving application behaviors and requirements.

Studying the sliding parameters of the window at the application level provides a thorough understanding of how different configurations impact predictive modeling in serverless computing. It provides a means to enhance these models, guaranteeing they are both feasible and effective in their predictive abilities.

5.3.5.3 Function-Level Analysis

The function level represents the most granular aspect of serverless computing, focusing on individual function invocations. This level of analysis, shown in Table 5.4, is essential for fine-tuning the performance of serverless systems at the most fundamental level. It is crucial to understand and optimize the invocation patterns of individual functions, which are the fundamental units of execution in serverless architectures. The comparative analysis includes scenarios with and without the application of dimensionality reduction, underscored the sensitivity of function invocations to windowing parameters, with distinct patterns emerging across different configurations.

Without PCA (Table 5.4a): At the function level, the non-PCA model delivers its best performance with an MAE of **12.3496**, utilizing a window size (W_z) of **120**, step size (S) of **15**, and target window size (W_y) of **30**. This setup indicates that, at the function level, a moderately sizeable historical context with a more considerable step size effectively captures the necessary temporal information for accurate predictions. The original (P_c) is **4037844** and the reduced data points (P_c') is **964653**, which reflect the volume of processed data with a notable reduction around **76.10%** indicating the efficiency of the windowing parameters chosen to condense the data while preserving essential information for prediction.

With PCA (Table 5.4b): By implementing PCA at the function level, the optimal window sliding configuration improves the MAE to **11.7049**. This performance is achieved with similar window sliding parameters as the optimal non-PCA configuration, demonstrating that PCA can enhance the

Table 5.4: Comparative Analysis of Window Sliding Parameters in Invocation Time Series Data at Function Level

(a) PCA = False

MAE	W_z	S	W_y	P_c	P_c'
12.3496	120	15	30	4037844	964653
12.5688	120	1	60	58525532	14108133
12.7044	120	1	30	59917892	13300193
13.1069	90	15	60	4037844	964653
13.4316	120	30	60	1995716	524678
13.5113	90	1	60	59917892	13300193
13.6413	120	15	60	3945020	1018527
13.8290	120	30	30	2042128	497655
14.2297	90	15	30	4130668	895434
14.3957	90	1	30	61310252	12259593
14.6203	60	15	60	4130668	895434
14.6787	90	30	60	2042128	497655
14.7083	90	30	30	2088540	462987
15.1948	60	30	60	2088540	462987
15.2487	60	1	60	61310252	12259593
15.7330	60	15	30	4223492	802729
16.2441	60	30	30	2134952	416190
16.6449	60	1	30	62702612	10892928
16.7448	30	15	60	4223492	802729
17.0246	30	30	60	2134952	416190
17.8540	30	1	60	62702612	10892928
19.2159	30	30	30	2181364	354307
19.5070	30	15	30	4316316	683458
21.3097	30	1	30	64094972	9220859

(b) PCA = True

MAE	W_z	S	W_y	P_c	P_c'
11.7049	120	15	30	4037844	964653
12.1630	120	30	60	1995716	524678
12.6766	120	15	60	3945020	1018527
12.7933	120	30	30	2042128	497655
12.8108	90	15	60	4037844	964653
12.9151	120	1	60	58525532	14108133
13.1676	120	1	30	59917892	13300193
13.6337	90	30	60	2042128	497655
13.6620	90	30	30	2088540	462987
13.9057	90	15	30	4130668	895434
14.0573	90	1	60	59917892	13300193
14.2341	60	30	60	2088540	462987
14.4291	60	15	60	4130668	895434
15.0695	90	1	30	61310252	12259593
15.3295	60	30	30	2134952	416190
15.6768	60	15	30	4223492	802729
15.8668	60	1	60	61310252	12259593
16.4722	30	30	60	2134952	416190
16.6106	30	15	60	4223492	802729
17.4837	60	1	30	62702612	10892928
18.1948	30	1	60	62702612	10892928
18.7374	30	30	30	2181364	354307
19.2839	30	15	30	4316316	683458
21.6151	30	1	30	64094972	9220859

model's predictive accuracy by concentrating on the most significant aspects of the data. The reduction in data volume is consistent with the non-PCA configuration, with PCA contributing to a more manageable and efficient predictive modeling process while ensuring the quality of predictions is improved.

Implications of Findings: The function-level comparative analysis sheds light on the importance of carefully selecting window sliding parameters to optimize predictive models in serverless environments. It demonstrates that larger window sizes can provide more historical context for predictions, but need to be balanced with the computational costs associated with processing more extensive data. The application of PCA shows promise in reducing these computational demands while maintaining or even enhancing predictive accuracy in certain configurations. However, it also highlights the need to carefully consider the amount of dimensionality reduction to ensure that

critical information is not lost.

Future Directions: Further research could explore adaptive and dynamic techniques for window sliding and dimensionality reduction at the function level, tailored to the unique characteristics and usage patterns of individual serverless functions. Investigating the integration of real-time data streams and incremental learning approaches could provide more accurate and up-to-date predictions. Such advances would contribute to more responsive and efficient serverless architectures that are capable of adapting to the evolving demands and behaviors of applications and users.

This function-level analysis is a critical component in understanding and optimizing serverless computing systems. By providing a nuanced view of the impact of window sliding parameters and dimensionality reduction techniques, this analysis contributes to the development of more sophisticated and effective predictive models for serverless function invocations.

5.3.5.4 Comparative Summary

The detailed findings from these comparative analyzes provide a wealth of information on the design and optimization of predictive models in serverless computing environments. At each level of analysis, as shown in Figures 5.2 and 5.3, the interaction between the windowing parameters and PCA highlights the multifaceted nature of predictive modeling, where the choice of parameters significantly influences the effectiveness of the model. Furthermore, the variation in MAE across different configurations sheds light on the inherent complexities of predicting serverless function invocations, underscoring the need for tailored approaches that take into account the unique characteristics of user, application, and function behaviors.

The heatmaps presented in Figure 5.2 provide a visual representation of the MAE for a LR model across different levels of analysis (user, application, and function) with varying window sizes (W_z), target sizes (W_y), and step sizes (S). The color gradients within the heatmaps range from blue to red, indicating low to high MAE values, respectively.

At the User level, the heatmaps contrast the effects of window sliding parameters on the MAE with and without PCA application. The heatmap without PCA shows that the lowest MAE is obtained with a larger W_z and smaller step size, suggesting that capturing finer temporal resolutions is crucial at this level. The application of PCA, as seen in the corresponding heatmap, generally results

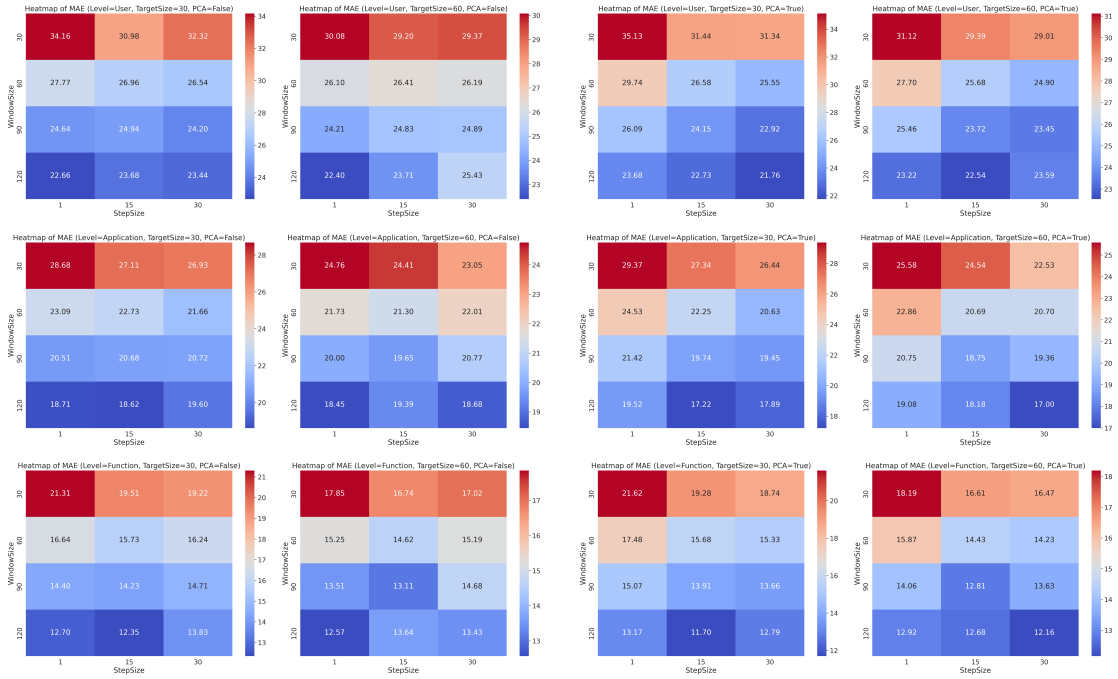

Figure 5.2: Comparative Heatmaps of Mean Absolute Error for Linear Regression Model: Effects of Window Size, Target Size, and Step Size Across User, Application, and Function Levels With and Without PCA.

in lower MAE values across all configurations, demonstrating the effectiveness of dimensionality reduction in enhancing model performance by filtering out noise and less relevant features. For the application level, the heatmap without PCA indicates that a larger historical context (larger W_z) tends to improve prediction accuracy, as lower MAE values are observed. Upon applying PCA, the heatmap shows an overall improvement in MAE, with the most significant reduction achieved with a W_z of **120** and a step size of **30**. This result highlights PCA's role in distilling critical features for improved predictive accuracy. The function level analysis heatmap without PCA suggests that a balance between W_z and step size is key to accurate predictions, with the lowest MAE recorded for an intermediate step size of 15. The introduction of PCA leads to an even lower range of MAE values, reinforcing the premise that dimensionality reduction, which focuses on preserving significant variance, is beneficial for model accuracy at the function level.

Across all levels, the introduction of PCA consistently enhances model accuracy, as evidenced by the cooler color tones in the heatmaps. Larger window sizes are typically associated with better performance, indicating the value of extensive historical data for the predictive models. The step

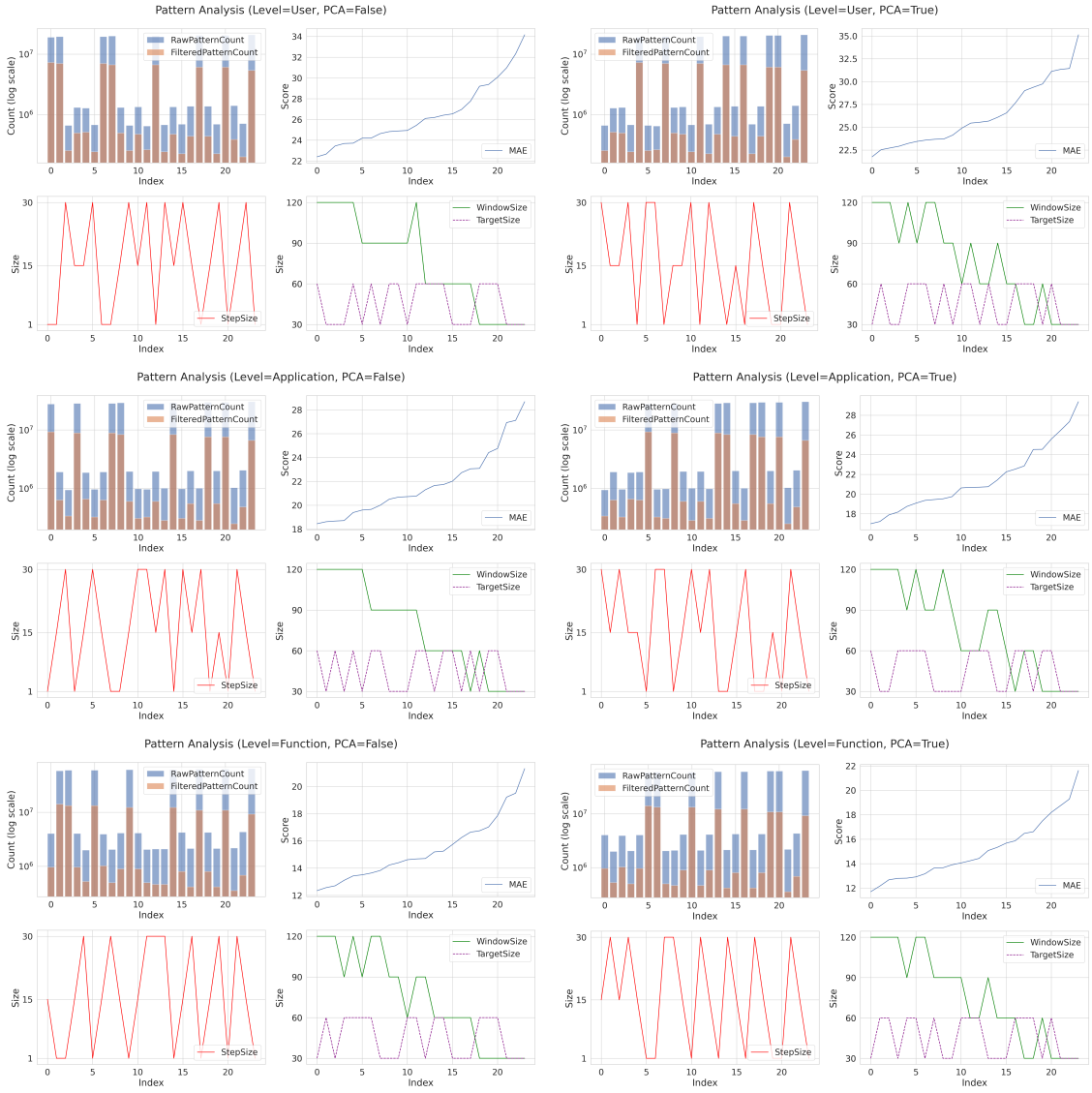

Figure 5.3: Comparison of Multi-Level Pattern Analysis using Linear Regression Model: Correlation Between Raw and Filtered Pattern Counts, Mean Absolute Error, and Windowing Parameter Variations, with and without PCA.

size needs to be optimized to avoid missing critical temporal patterns or failing to capture sufficient data variability. The heatmaps effectively guide the selection of optimal window sliding parameters for serverless computing predictive models. They emphasize the trade-off between the capture of detailed temporal information and the computational gains achieved through dimensionality reduction techniques such as PCA.

Figure 5.3 provides a comprehensive comparison of pattern analysis at multiple levels, highlighting the correlation between raw (P_c) and filtered (P'_c) pattern counts, MAE, and the variations in windowing parameters with and without the application of PCA. A critical aspect of this analysis is the emphasis on reducing pattern counts. The bar graphs illustrate the comparison between raw pattern counts and filtered pattern counts after redundant patterns are removed. This reduction is crucial as it directly influences the MAE, with a significant reduction in patterns often correlating with improved predictive accuracy, as indicated by the line graphs showing the MAE trend.

At the user level, the graphs reveal that without PCA, while there is a substantial reduction in pattern count, the MAE remains relatively high. With PCA, not only does the pattern count reduction remain significant, but the MAE also decreases, underscoring PCA role in enhancing performance by focusing on the most informative features. Similarly, application-level graphs demonstrate a notable reduction in pattern count. It is observed that the application of PCA contributes to a further reduction in MAE, indicating an efficient balance between data simplification and preservation of predictive quality. At the function level, the reduction in pattern count is consistent with the other levels, and the impact of PCA is again apparent, with a reduction in MAE, emphasizing the effectiveness of PCA in yielding a concise yet powerful set of characteristics for prediction.

Line graphs showing window size, target size, and step size variations offer insight into how different configurations affect the model. A larger window size often results in a lower MAE, but the optimal configuration also depends on the appropriate combination of target size and step size. The reduction in pattern count at all levels, particularly when PCA is applied, plays a significant role in improving the predictive modeling process. This analysis underscores the importance of careful parameter tuning and the effectiveness of dimensionality reduction techniques in optimizing serverless computing predictive models.

5.3.6 Model Performance Evaluation

5.3.6.1 Learning Curves

Figure 5.4 presents the learning curves of the LR model's performance at each analysis level. These curves are plotted to compare the training and testing MAE, R^2 score, and MSE against the

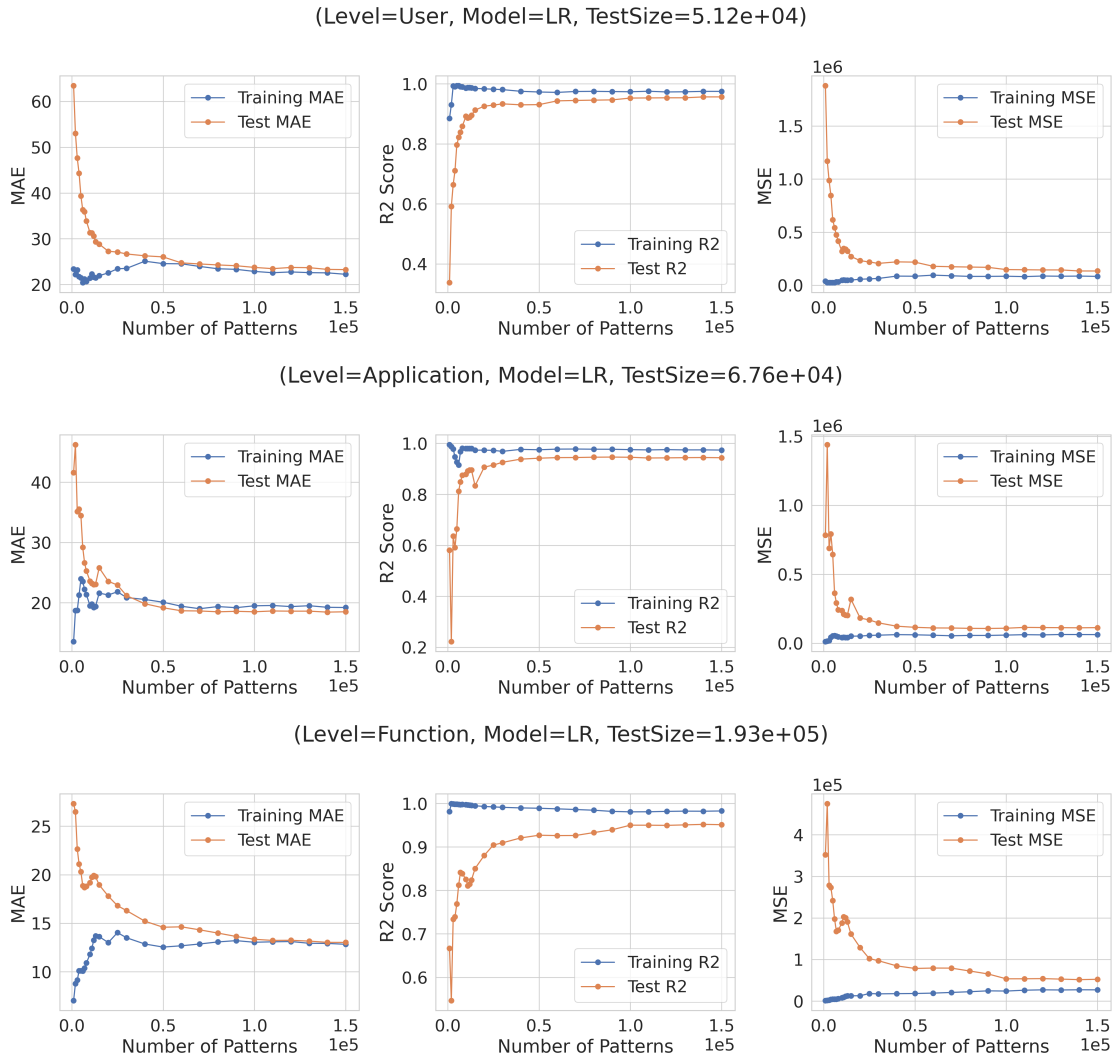

Figure 5.4: Learning Curves of Linear Regression Model Performance at User, Application, and Function Levels: Comparison of Training and Testing MAE, R2 Score, and MSE Against the Number of Patterns.

number of patterns used in the time series data. The time series data for each level has been prepared based on the best-performing settings, which achieved the lowest MAE based on previous findings, thus ensuring an optimized learning process.

The learning curves are indicative of the model's ability to learn from a given number of patterns. A decrease in MAE and MSE, along with an increase in the R^2 score for both the training and the test data sets as the number of patterns increases, suggests that the model effectively captures the underlying trends and dynamics of the data. It is evident from the curves that as more data is

provided, the model's performance on the test set converges towards its performance on the training set, which is a desirable trait indicating good generalization.

The windowing operation, an essential feature of time series preprocessing, has been used to improve the learning process. By systematically selecting the best window configurations for each level, the model is better able to capture the temporal dependencies and nuances within the data. This operation improves the learning ability of the model by providing it with a structured format of input data, which is particularly important for the regression models used in this study.

In summary, the learning curves underscore the importance of careful time series data preparation and the positive impact of the windowing operation on the model's learning process. The convergence of the training and testing curves as the number of patterns increases is a positive sign that the model is able to generalize well to new data. This finding means that the model is not overfitting to the training data and can make accurate predictions on unseen data.

5.3.6.2 Performance Comparison of Regression Models on Test Data

We present a detailed evaluation of various regression models on test data at the three distinct levels. Table 5.5 displays the comparative performance metrics for each model, with a particular emphasis on the best-performing settings based on the MAE.

At the User level, as shown in Table 5.5a, the KNN model outperforms the others with the lowest MAE of **19.19**, indicating its superior accuracy in capturing user behavior within the serverless environment. The ETR and RFR models also show commendable performance, with only slight differences in MAE, MSE, and R^2, suggesting their robustness in handling user-level data. The LR model, while not outperforming the ensemble methods, still maintains a competitive R^2 score, highlighting its effectiveness as a simpler alternative. The DTR model exhibits the highest MAE and MSE, indicating a relative underperformance in this context.

Moving to the application level, as detailed in Table 5.5b, we observe a similar pattern with KNN achieving the lowest MAE of **15.02**. The ETR and RFR models closely follow, with marginal differences in MAE and MSE values but comparable R^2 scores. The LR model, despite its simplicity, presents a reasonable MAE and R^2, underscoring its utility in application-level predictions. The DTR model, however, shows a notable decrease in performance, reflected by the highest MAE and

Table 5.5: Evaluation of Regression Models on Test Data: Comparative Performance Metrics

(a) Level = User

Model	MAE	MSE	R^2
KNN	19.19	124893	0.9599
ET	19.83	102214	0.9663
RF	20.39	100046	0.9668
LR	21.76	118172	0.9615
DT	26.18	164078	0.9455

(b) Level = Application

Model	MAE	MSE	R^2
KNN	15.02	105949	0.9472
ET	15.97	104460	0.9477
RF	16.20	105512	0.9471
LR	17.00	107271	0.9466
DT	21.88	188880	0.9038

(c) Level = Function

Model	MAE	MSE	R^2
ET	9.71	41832	0.9615
KNN	9.89	53004	0.9519
RF	10.07	44069	0.9596
LR	11.70	48005	0.9558
DT	12.87	79606	0.9268

the lowest R^2 score among the evaluated models.

At the Function level, Table 5.5c indicates that the ETR model secures the best MAE of **9.71**, closely followed by the KNN and RFR models. These models exhibit strong predictive capabilities, as evidenced by their R^2 scores exceeding **0.95**. The LR model, while slightly lagging in MAE, maintains a R^2 score above **0.95**, suggesting its adequacy for function-level prediction tasks. The DTR model shows the largest discrepancy in MAE and the lowest R^2 score, implying a less precise fit to the function-level data compared to its counterparts.

For each level, the time series data is meticulously prepared based on the configuration that achieved the lowest MAE in the comparative analysis of the window sliding parameters. This approach ensures that the models are evaluated on data that is optimized for their learning algorithms, providing a fair and rigorous assessment of their predictive performance.

The evaluation of regression models on the test data demonstrates the varying effectiveness of different modeling approaches at the user, application, and function levels. The use of best-performing settings for data preparation has proven to be a decisive factor in enhancing model accuracy, as indicated by the MAE across all levels. The results provide valuable information for selecting appropriate models for serverless computing environments based on the specific requirements of each analytical level.

5.3.6.3 Evaluation of Regression Models Over Time

Table 5.6 provides an aggregated evaluation of the performance of different regressor models, analyzing the mean score of key metrics over consecutive days. This longitudinal analysis assesses the stability and reliability of the models at each analysis level.

Table 5.6: Evaluation of Regressor Performance: Mean Score Analysis of Key Metrics Over Consecutive Days

(a) Level = User

Model	MAE	MSE	R^2
KNN	21.53	206950	0.9403
ET	21.86	205178	0.9408
LR	22.57	196447	0.9428
RF	23.21	222022	0.9361
DT	29.94	330151	0.9045

(b) Level = Application

Model	MAE	MSE	R^2
KNN	17.58	146096	0.9383
ET	18.13	142008	0.9401
LR	18.59	137825	0.9412
RF	19.27	156298	0.9343
DT	25.47	265250	0.8877

(c) Level = Function

Model	MAE	MSE	R^2
ET	11.99	85568	0.9401
KNN	12.07	87829	0.9386
RF	12.57	90116	0.9369
LR	12.85	82413	0.9418
DT	15.86	131258	0.9080

At the user level, Table 5.6a shows that the KNN model achieves the lowest mean MAE, indicating its strength to consistently predict user behavior with minimal deviation. The ETR and LR models closely follow, with slightly higher MAE values but better MSE performance, suggesting their efficiency in minimizing error across predictions. The RFR model, while exhibiting a higher MAE and MSE, still maintains a satisfactory R^2 score. The DTR model, with the highest MAE and MSE, shows the most significant variation in the predictions over time.

In the context of the application level, as depicted in Table 5.6b, the KNN model again presents the lowest mean MAE, endorsing its robustness in application-level predictions. The ETR model shows comparable performance with a slightly higher MAE but a lower MSE, while the LR model scores the best in terms of R^2, indicating a strong correlation with the observed data. The RFR model records a modest increase in mean MAE and MSE, and the DTR model ranks last with the highest mean errors scores, pointing to less consistency in its predictions.

For the function level, detailed in Table 5.6c, the ETR model achieves the lowest mean MAE, reinforcing its effectiveness in function-level forecasting. KNN and RFR models also perform well, maintaining mean MAE scores within a close range. The LR model, despite a slightly higher MAE, attains the best R^2 score, suggesting that its predictive accuracy is quite reliable. The DTR model, as observed at other levels, has the highest mean MAE and MSE, indicating a broader variability in its daily predictions.

This comparative performance analysis over consecutive days highlights the importance of model selection based on consistent performance metrics. Although some models excel in certain metrics, a comprehensive view of all scores is crucial for selecting a model that offers reliability and consistency in a serverless computing environment.

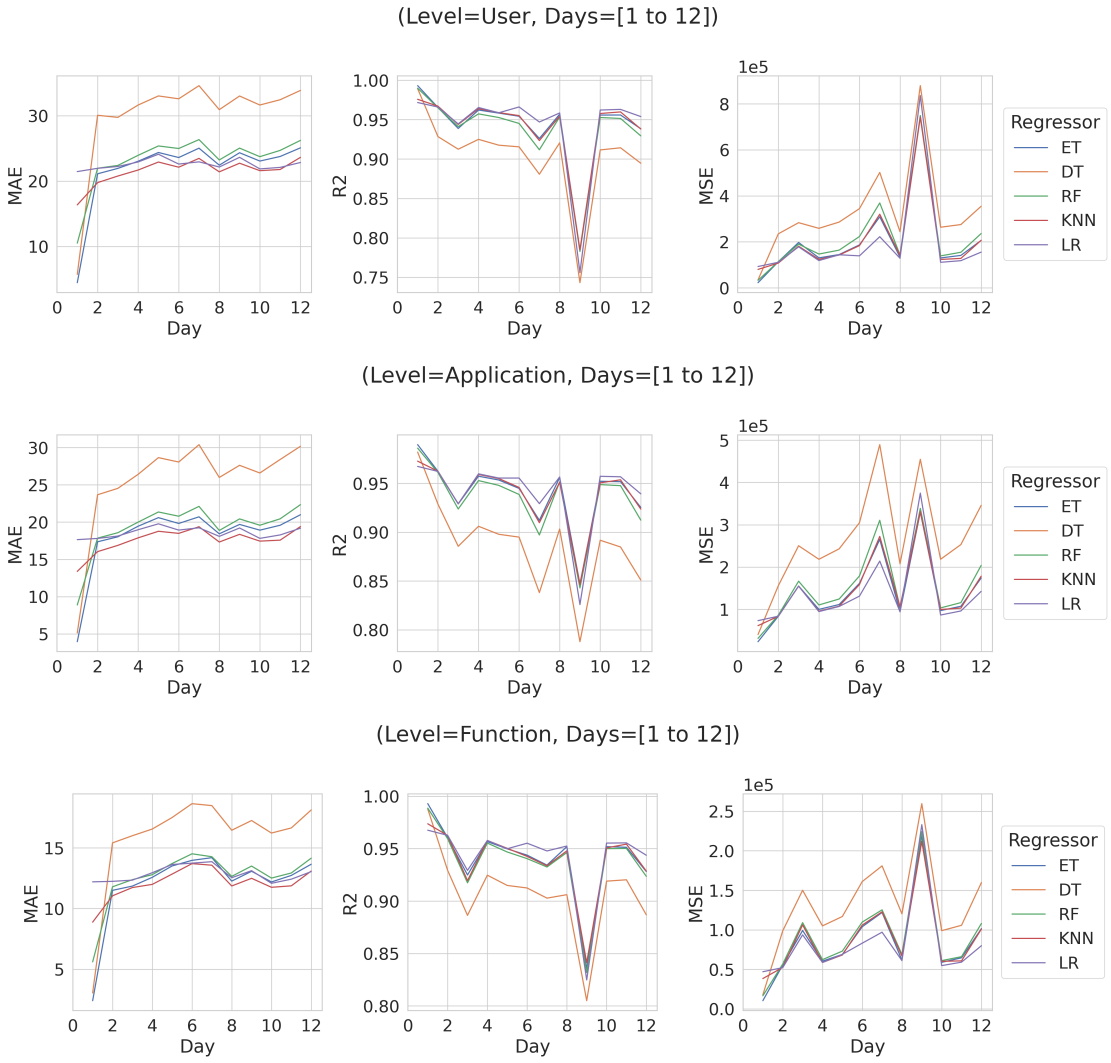

Figure 5.5: Daily Performance Trends of Various Regressors at User, Application, and Function Levels: Comparative Analysis of MAE, R2 Score, and MSE Over a 12-Day Period.

5.3.6.4 Stability Analysis of Regression Models

Figure 5.5 illustrates the daily performance trends of various regression models over a 12 day period, at the user, application, and function levels. This analysis provides insights into the consistency and variability of model predictions over time, highlighting the reliability of each regression approach in a dynamic serverless computing environment.

The MAE, R^2 score, and MSE for each type of model (ETR, DTR, RFR, KNN, and LR) are

plotted daily. Trends show the extent to which each model is able to maintain consistency of performance on successive days. A stable MAE and a consistently high R^2 score are desirable, indicating that the model's predictions are accurate and reliable over time. The MSE gives a sense of the prediction error's variance; lower values denote more precise predictions. For each level, the stability of the models is crucial for ensuring accurate predictions of serverless function invocations, which are integral to resource allocation and management within serverless architectures.

The comparative analysis of MAE, R^2, and MSE reveals the stability of the daily performance of the regressors, which informs the selection of the most robust model for each level of analysis. It is evident that some models exhibit greater variability in their performance metrics, while others maintain a more consistent trend. This detailed daily performance stability analysis is instrumental in understanding the temporal dynamics of the model predictions. The findings of this analysis are critical for determining the drop performance threshold and, accordingly, inform the redevelopment of the model with the recent data for more reliable and stable predictive models in serverless computing environments.

Figure 5.6 presents a boxplot summary of the daily performance effectiveness of various regressors and metrics over 12 days for each level. Boxplots are utilized to represent the distribution of the performance metrics, providing insight into the median, interquartile range, and outliers of each regressor's performance over the observed period. The central line of each box represents the median value, while the top and bottom edges indicate the $75th$ and $25th$ percentiles, respectively. Outliers are depicted as individual points beyond the whiskers of the boxplots.

For each level, the boxplots convey the variability and central tendency in MAE, R^2, and MSE for each regressor. The distribution width signifies the stability of the model's performance. The narrower boxes suggest consistent performance, while the wider ones indicate variability over days. The MAE and MSE boxplots help to understand the magnitude and variance of the error, while the R^2 boxplots indicate the consistency of the predictive accuracy of the model. The comparative size of the boxes and the position of the median line provide an immediate visual cue about the model's performance over the examined days. These boxplot summaries are instrumental in evaluating the daily performance efficacy of regressors. They enable a quick assessment of which models are more stable and reliable over time, which is paramount for tasks requiring consistent predictive

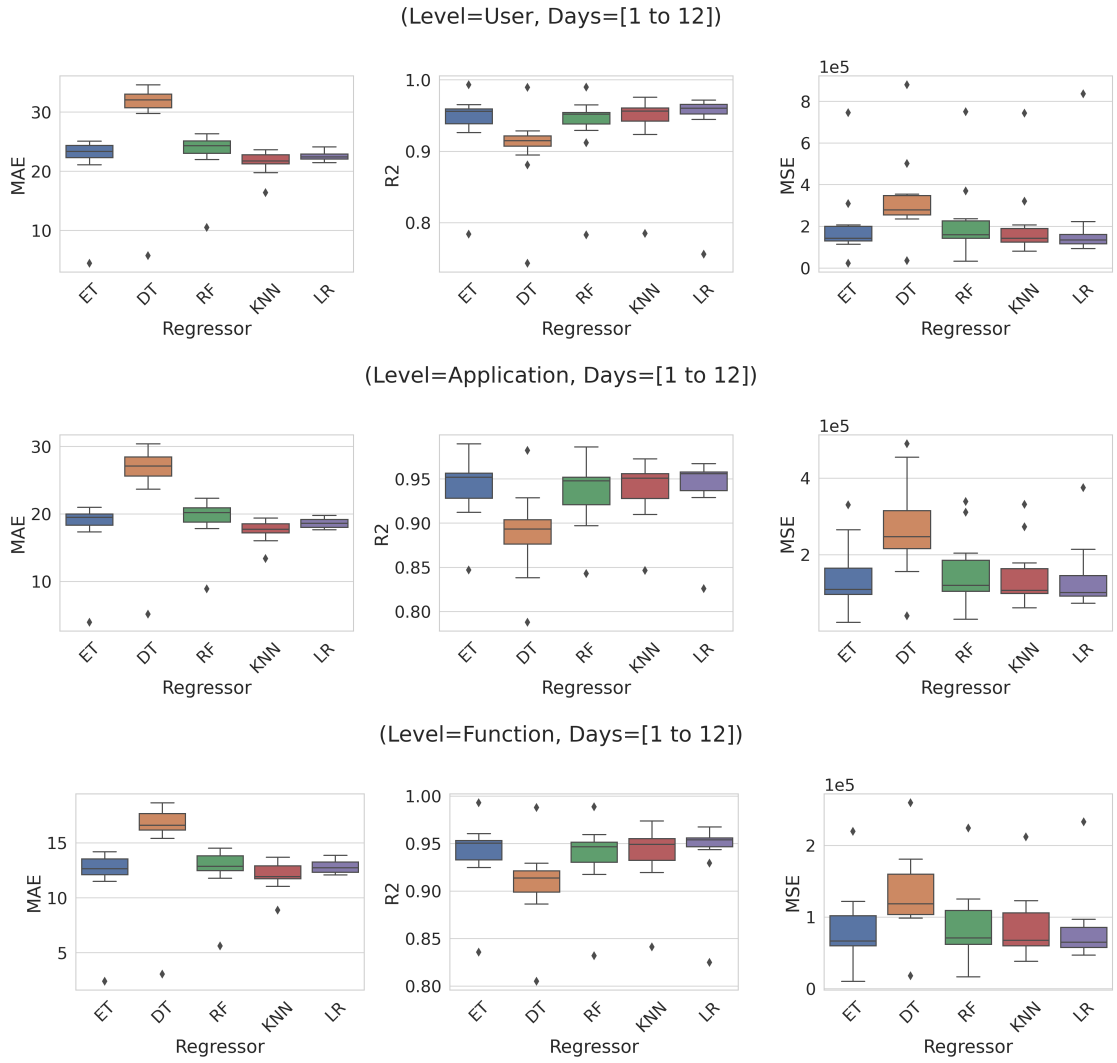

Figure 5.6: Boxplot Summary of Daily Performance Efficacy of Various Regressors Over 12 Days: Comparing MAE, R2 Score, and MSE Across User, Application, and Function Levels.

performance in a serverless computing environment. They provide a comprehensive overview of the performance variations and stability of different regression models over consecutive days. This graphical summary is essential for selecting the most appropriate model based on the specific needs of user, application, and function levels within serverless architectures.

By synthesizing these insights, the study contributes to a deeper understanding of the dynamics at play in serverless computing environments. It offers a set of empirically grounded recommendations for practitioners looking to implement predictive models in such settings, emphasizing the importance of customizing the model configuration to fit the specific needs and constraints of the

environment. Furthermore, the findings lay a foundation for future research, suggesting areas for further exploration, such as the development of adaptive models that can dynamically adjust their configuration in response to changing patterns of usage or the integration of additional data sources to enrich the predictive capability of the models. As serverless computing continues to evolve, the insights derived from this comprehensive analysis will play a crucial role in guiding the development of more sophisticated, efficient, and accurate predictive models. By continually refining these models, we can better anticipate and respond to the demands of serverless environments, ultimately leading to more robust, scalable, and user-responsive computing solutions.

5.4 Conclusion

This study has presented a comprehensive exploration into enhancing the predictability and efficiency of function invocations in serverless computing environments, and was submitted in [138]. Through a systematic approach employing multi-output regression models, windowing techniques, and PCA for dimensionality reduction, we have provided valuable insights and methodologies that push forward the capabilities of serverless computing.

(1) Our multilevel predictive modeling has demonstrated significant potential in understanding and predicting function invocation patterns across user, application, and function levels. This granular approach is pivotal for fine-tuning resource allocation and improving operational efficiency in cloud environments.

(2) The detailed exploration of windowing techniques and the strategic application of PCA have revealed the importance of optimizing data preprocessing and feature extraction in predictive modeling. Our findings emphasize the balance between maintaining data integrity and computational efficiency, a critical consideration in large-scale data environments.

(3) The development of a comparative analysis framework and the utilization of a real-world cloud workload trace have allowed for a thorough evaluation of model performances. This framework is instrumental in identifying optimal configurations and ensuring that the predictive models are not only theoretically sound but also practically viable.

(4) Furthermore, the assessment of temporal stability and performance variations of the models over consecutive days contributes to the reliability and robustness of predictive systems in serverless computing, addressing a significant challenge in the field.

(5) Lastly, our research has outlined several pathways for future work, encouraging continued advancements in the predictive modeling of serverless computing workloads.

The contributions of this study are intended to serve as a foundation for future research and practical applications in serverless computing. We advocate for continued exploration and innovation in this domain, as the accurate prediction of function invocations is paramount in optimizing cloud resources and enhancing service delivery. As serverless architectures evolve, so too must the methodologies and tools designed to support them, ensuring that they remain efficient, cost-effective, and responsive to the needs of diverse applications.

Chapter 6

Conclusion and Future Work

6.1 Concluding Remarks

This thesis represents a comprehensive investigation into the landscape of cloud computing, with particular emphasis on developing and refining a methodological framework that integrates advanced clustering and predictive modeling techniques. Throughout this extensive exploration, crucial insights have been unearthed, and innovative strategies have been devised to enhance the comprehension and management of cloud workloads and serverless computing environments.

In Chapter 3, we introduced advanced clustering techniques for the workload segmentation process within cloud data centers, highlighting the unique attributes and suitability of different data pipelines. Additionally, we presented an advanced concept of ensemble clustering that combines multiple clustering methods incorporated in various data pipelines for a more comprehensive and nuanced analysis of cloud workloads. This innovative strategy is particularly pertinent to address the multifaceted challenges posed by the dynamic nature of cloud environments, ensuring more efficient and effective monitoring and workload management.

In Chapter 4, we introduced various methodologies and innovations for single-output predictive modeling within cloud environments. Initially, we presented an advanced multilevel learning model incorporating anomaly, clustering, and ensemble learning methods to provide high-precision prediction for CPU utilization in cloud data centers. Then, the economic aspects of cloud computing are explored through a regression-based approach for proactive price predictive modeling, which helps

navigate the complexities of cloud service pricing models. Finally, we explored the use of combined imbalance and ensemble learning methods to improve load prediction in cloud computing systems, a key factor in ensuring efficient resource management in dynamic cloud environments. Collectively, we provide a better understanding of various predictive modeling methods in cloud data centers and demonstrate their practical applications.

In Chapter 5, we further expanded the horizon of predictive modeling within the cloud environment by delving into multi-output prediction for serverless computing. Using multi-output regression models, windowing techniques, and Principal Component Analysis (PCA) for dimensionality reduction has offered invaluable insights into function invocation patterns at various levels of analysis. This approach has improved predictive accuracy and emphasized the critical balance between data integrity and computational efficiency, a cornerstone of large-scale data processing.

The research encompassed in this thesis yields substantial contributions to the field of cloud computing. Specifically, it provides a robust foundation for both theoretical comprehension and practical application with respect to optimizing prediction precision in various aspects within cloud environments. Therefore, the findings presented in this thesis have immense value for professionals, researchers, and decision makers operating within the cloud domain.

6.2 Future Work

Our forthcoming main task involves the application of a proposed methodological framework that amalgamates advanced clustering and predictive modeling techniques in practical cloud environments. The objective is to predict which actions and resource reconfigurations can be applied proactively in operational management. Integrating this framework will enable the system to operate with greater efficiency. However, in the future, several avenues for future research and development can further augment the contributions of this thesis.

In the context of workload segmentation, a range of ensemble clustering techniques should be investigated, specifically those that enable the fine-tuning of the preprocessing pipeline to accommodate the dynamic nature of cloud workloads. In addition, an in-depth analysis of the practicality of

deploying the proposed clustering methodology across multiple cloud workloads could yield valuable insights into its comprehensive applicability and resilience. This examination would provide a more nuanced understanding of the clustering approach's potential to deliver robust and universal solutions in various cloud environments.

In single-output predictive modeling, future work could improve the proposed models by using hybrid learning models considering various prediction aspects within the cloud data centers. Enhancing model robustness and precision is critical to optimizing various cloud workloads. Therefore, in future work, it is essential also to include diverse datasets from various cloud providers that can cater to the complexities associated with the dynamic nature of cloud-based workloads.

In multi-output predictive modeling, there is a significant opportunity to expand the prediction techniques to encompass a broader range of aspects within cloud environments. Investigating the applicability of the proposed methodologies in different cloud settings beyond the function invocation in serverless computing would help assess their universal applicability and robustness.

Furthermore, continuous advancements in cloud computing technology necessitate ongoing research to ensure that predictive models and clustering techniques remain efficient, cost-effective, and responsive to the needs of various applications. It includes technological improvements and a focus on enhancing the user experience and the economic aspects of cloud computing, such as cost estimation and resource optimization. In conclusion, this thesis sets the stage for ongoing innovation in cloud computing, encouraging continued exploration and development in this dynamic and ever-important field. The methodologies and findings presented here are poised to significantly contribute to optimizing cloud resources, ultimately leading to improved business results and customer experiences in the cloud computing landscape.

Bibliography

[1] W. Venters and E. A. Whitley, "A critical review of cloud computing: Researching desires and realities," *Journal of Information Technology*, vol. 27, pp. 179–197, 2012. DOI: 10.1057/jit.2012.17.

[2] M. H. Shirvani, A. M. Rahmani, and A. Sahafi, "A survey study on virtual machine migration and server consolidation techniques in dvfs-enabled cloud datacenter: Taxonomy and challenges," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, pp. 267–286, 3 Mar. 2020, ISSN: 13191578. DOI: 10.1016/j.jksuci.2018.07.001. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1319157818302842.

[3] M. Ala'Anzy and M. Othman, "Load balancing and server consolidation in cloud computing environments: A meta-study," *IEEE Access*, vol. 7, pp. 141 868–141 887, 2019. DOI: 10.1109/ACCESS.2019.2944420.

[4] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of network and computer applications*, vol. 52, pp. 11–25, 2015. DOI: 10.1016/j.jnca.2015.02.002.

[5] S. Mustafa, B. Nazir, A. Hayat, A. ur Rehman Khan, and S. A. Madani, "Resource management in cloud computing: Taxonomy, prospects, and challenges," *Computers & Electrical Engineering*, vol. 47, pp. 186–203, 2015, ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2015.07.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004579061500275X.

[6] D. Soni and N. Kumar, "Machine learning techniques in emerging cloud computing integrated paradigms: A survey and taxonomy," *Journal of Network and Computer Applications*, vol. 205, p. 103 419, 2022, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103419. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522000765.

[7] M. Armbrust, A. Fox, R. Griffith, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[8] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International journal of computer science*, vol. 1, no. 2, pp. 111–117, 2006.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[11] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[12] N. Thakur, A. Singh, and A. Sangal, "Cloud services selection: A systematic review and future research directions," *Computer Science Review*, vol. 46, p. 100 514, 2022.

[13] K. Zaman, A. Hussain, M. Imran, and M. Sohail, "Cost-effective data replication mechanism modelling for cloud storage," *International Journal of Grid and Utility Computing*, vol. 13, no. 6, pp. 652–669, 2022.

[14] S. Jayaprakash, M. D. Nagarajan, R. P. d. Prado, S. Subramanian, and P. B. Divakarachari, "A systematic review of energy management strategies for resource allocation in the cloud: Clustering, optimization and machine learning," *Energies*, vol. 14, no. 17, p. 5322, 2021.

[15] L. Zhu, K. Huang, K. Fu, Y. Hu, and Y. Wang, "A priority-aware scheduling framework for heterogeneous workloads in container-based cloud," *Applied Intelligence*, vol. 53, no. 12, pp. 15 222–15 245, 2023.

[16] Q. Xia, Y. Lan, L. Zhao, and L. Xiao, "Energy-saving analysis of cloud workload based on k-means clustering," in *2014 IEEE Computers, Communications and IT Applications Conference*, IEEE, 2014, pp. 305–309.

[17] R. Estrada, I. Valeriano, and X. Aizaga, "Cpu usage prediction model: A simplified vm clustering approach," in *Conference on Complex, Intelligent, and Software Intensive Systems*, Springer, 2023, pp. 210–221.

[18] A. Katal, S. Dahiya, and T. Choudhury, "Workload characterization and classification: A step towards better resource utilization in a cloud data center.," *Pertanika Journal of Science & Technology*, vol. 31, no. 5, 2023.

[19] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: A hybrid approach," *Cluster Computing*, vol. 24, no. 1, pp. 319–342, 2021.

[20] M. Askarizade Haghighi, M. Maeen, and M. Haghparast, "An energy-efficient dynamic resource management approach based on clustering and meta-heuristic algorithms in cloud computing iaas platforms: Energy efficient dynamic cloud resource management," *Wireless Personal Communications*, vol. 104, pp. 1367–1391, 2019.

[21] N. Dezhabad, S. Ganti, and G. Shoja, "Cloud workload characterization and profiling for resource allocation," in *2019 IEEE 8th international conference on cloud networking (CloudNet)*, IEEE, 2019, pp. 1–4.

[22] P. Neamatollahi, S. Abrishami, M. Naghibzadeh, M. H. Y. Moghaddam, and O. Younis, "Hierarchical clustering-task scheduling policy in cluster-based wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 1876–1886, 2017.

[23] P. Orzechowski, J. Proficz, H. Krawczyk, and J. Szymański, "Categorization of cloud workload types with clustering," in *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*, D. K. Lobiyal, D. P. Mohapatra, A. Nagar, and M. N. Sahoo, Eds., Springer, New Delhi: Springer India, 2017, pp. 303–313, ISBN: 978-81-322-3592-7.

[24] A. Jivrajani, D. Raghu, K. Apoorva, H. Phalachandra, and D. Sitaram, "Workload characterization and green scheduling on heterogeneous clusters," in *2016 22nd Annual International Conference on Advanced Computing and Communication (ADCOM)*, IEEE, 2016, pp. 3–8.

[25] S. A. Yousif and A. Al-Dulaimy, "Clustering cloud workload traces to improve the performance of cloud data centers," in *Proceedings of the World Congress on Engineering*, vol. 1, 2017, pp. 7–10.

[26] Z. Gu, S. Tang, B. Jiang, S. Huang, Q. Guan, and S. Fu, "Characterizing job-task dependency in cloud workloads using graph learning," in *2021 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, IEEE, 2021, pp. 288–297.

[27] J. Gao, H. Wang, and H. Shen, "Machine learning based workload prediction in cloud computing," in *2020 29th international conference on computer communications and networks (ICCCN)*, IEEE, 2020, pp. 1–9.

[28] S. Ismaeel, A. Al-Khazraji, and A. Miri, "An efficient workload clustering framework for large-scale data centers," in *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, IEEE, 2019, pp. 1–5.

[29] P. Bhattacharjee and P. Mitra, "A survey of density based clustering algorithms," *Frontiers of Computer Science*, vol. 15, pp. 1–27, 2021.

[30] Y. Liang, K. Chen, L. Yi, X. Su, and X. Jin, "Degtec: A deep graph-temporal clustering framework for data-parallel job characterization in data centers," *Future Generation Computer Systems*, vol. 141, pp. 81–95, 2023.

[31] H. Ikhlasse, D. Benjamin, C. Vincent, and M. Hicham, "Multimodal cloud resources utilization forecasting using a bidirectional gated recurrent unit predictor based on a power efficient stacked denoising autoencoders," *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 11 565–11 577, 2022.

[32] S. S. Gill, S. Tuli, A. N. Toosi, *et al.*, "Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments," *Journal of Systems and Software*, vol. 166, p. 110 596, 2020.

[33] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An efficient deep learning model to predict cloud workload for industry informatics," *IEEE transactions on industrial informatics*, vol. 14, no. 7, pp. 3170–3178, 2018.

[34] S. Gupta, N. Muthiyan, S. Kumar, A. Nigam, and D. A. Dinesh, "A supervised deep learning framework for proactive anomaly detection in cloud workloads," in *2017 14th IEEE India Council International Conference (INDICON)*, IEEE, 2017, pp. 1–6.

[35] Q. Yang, Y. Zhou, Y. Yu, J. Yuan, X. Xing, and S. Du, "Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing," *The Journal of Supercomputing*, vol. 71, pp. 3037–3053, 2015.

[36] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal of machine learning research*, vol. 3, no. 12, pp. 583–617, 2002.

[37] A. Topchy, A. K. Jain, and W. Punch, "A mixture model for clustering ensembles," in *Proceedings of the 2004 SIAM international conference on data mining*, SIAM, 2004, pp. 379–390.

[38] R. Caruana, M. Elhawary, N. Nguyen, and C. Smith, "Meta clustering," in *Sixth International Conference on Data Mining (ICDM'06)*, IEEE, 2006, pp. 107–118.

[39] B. Zhou, B. Lu, and S. Saeidlou, "A hybrid clustering method based on the several diverse basic clustering and meta-clustering aggregation technique," *Cybernetics and Systems*, pp. 1–27, 2022.

[40] K. Li, X. Cao, X. Ge, *et al.*, "Meta-heuristic optimization-based two-stage residential load pattern clustering approach considering intra-cluster compactness and inter-cluster separation," *IEEE Transactions on Industry Applications*, vol. 56, no. 4, pp. 3375–3384, 2020.

[41] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, pp. 241–258, 2020.

[42] K. L. Devi and S. Valli, "Time series-based workload prediction using the statistical hybrid model for the cloud environment," *Computing*, vol. 105, no. 2, pp. 353–374, 2023.

[43] S. Sarikaa, S. Niranjana, *et al.*, "Time series forecasting of cloud resource usage," in *2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)*, IEEE, 2021, pp. 372–382.

[44] J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," *Computer Communications*, 2023.

[45] S. Kashyap and A. Singh, "Prediction-based scheduling techniques for cloud data center's workload: A systematic review," *Cluster Computing*, pp. 1–27, 2023.

[46] M. Dubrovin, I. Gluhih, and I. Karyakin, "Forecasting the server status using the triple exponential smoothing model," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1661, 2020, p. 012 031.

[47] A. C. Caminero, S. Ros, R. Hernández, A. Robles-Gómez, and R. Pastor, "Cloud-based e-learning infrastructures with load forecasting mechanism based on exponential smoothing: A use case," in *2011 Frontiers in Education Conference (FIE)*, IEEE, 2011, S3C–1.

[48] Z. Zou, Y. Xie, K. Huang, G. Xu, D. Feng, and D. Long, "A docker container anomaly monitoring system based on optimized isolation forest," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 134–145, 2019.

[49] R. N. Calheiros, K. Ramamohanarao, R. Buyya, C. Leckie, and S. Versteeg, "On the effectiveness of isolation-based anomaly detection in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 18, e4169, 2017. DOI: https://doi.org/10.1002/cpe.4169. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4169.

[50] P. Ntambu and S. A. Adeshina, "Machine learning-based anomalies detection in cloud virtual machine resource usage," in *2021 1st International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS)*, IEEE, 2021, pp. 1–6.

[51] S. Bansal, A. Singh, S. Bijlwan, B. Goyal, A. Dogra, and D. C. Lepcha, "Cloud computing improved clustering using intrusion detection," in *2023 International Conference in Advances in Power, Signal, and Information Technology (APSIT)*, IEEE, 2023, pp. 1–5.

[52] B. Agrawal, T. Wiktorski, and C. Rong, "Adaptive real-time anomaly detection in cloud infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, e4193, 2017.

[53] G. Mahesh and V. Parthipan, "A novel approach for statistical analysis of cpu utilization in cloud computing using graphic clustering algorithm with hierarchical clustering algorithm based on accuracy and root mean square error (rsme) value," in *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, 2022, pp. 1315–1320. DOI: 10.1109/ICSCDS53736.2022.9760766.

[54] Z. Ma, D. Ma, M. Lv, and Y. Liu, "Virtual machine migration techniques for optimizing energy consumption in cloud data centers," *IEEE Access*, 2023.

[55] I. Nisce, X. Jiang, and S. P. Vishnu, "Machine learning based thermal prediction for energy-efficient cloud computing," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, IEEE, 2023, pp. 624–627.

[56] S. Parthasarathy, "Osvr: An efficient support vector regression model based host overload detection and secure virtual machine migration," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 6, pp. 7309–7317, 2023.

[57] W. Zhong, Y. Zhuang, J. Sun, and J. Gu, "A load prediction model for cloud computing using pso-based weighted wavelet support vector machine," *Applied Intelligence*, vol. 48, pp. 4072–4083, 2018.

[58] L. M. Al Qassem, T. Stouraitis, E. Damiani, and I. A. M. Elfadel, "Proactive random-forest autoscaler for microservice resource allocation," *IEEE Access*, vol. 11, pp. 2570–2585, 2023.

[59] Z. Zhou, M. Shojafar, M. Alazab, and F. Li, "Iecl: An intelligent energy consumption model for cloud manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 8967–8976, 2022.

[60] R. S. Thakkar, D. Thakkar, and M. Bhavsar, "Mvms: Rnn based pro-active resource scaling in cloud environment," *Scalable Computing: Practice and Experience*, vol. 24, no. 1, pp. 17–33, 2023.

[61] M. Duggan, K. Mason, J. Duggan, E. Howley, and E. Barrett, "Predicting host cpu utilization in cloud computing using recurrent neural networks," in *2017 12th international conference for internet technology and secured transactions (ICITST)*, IEEE, 2017, pp. 67–72.

[62] K. Valarmathi and S. Kanaga Suba Raja, "Resource utilization prediction technique in cloud using knowledge based ensemble random forest with lstm model," *Concurrent Engineering*, vol. 29, no. 4, pp. 396–404, 2021.

[63] N. Tran, T. Nguyen, B. M. Nguyen, and G. Nguyen, "A multivariate fuzzy time series resource forecast model for clouds using lstm and data correlation analysis," *Procedia Computer Science*, vol. 126, pp. 636–645, 2018.

[64] D. Saxena, J. Kumar, A. K. Singh, and S. Schmid, "Performance analysis of machine learning centered workload prediction models for cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1313–1330, 2023.

[65] Y. Lu, L. Liu, J. Panneerselvam, X. Zhai, X. Sun, and N. Antonopoulos, "Latency-based analytic approach to forecast cloud workload trend for sustainable datacenters," *IEEE Transactions on Sustainable Computing*, vol. 5, pp. 308–318, 3 Jul. 2020, ISSN: 23773782. DOI: 10.1109/TSUSC.2019.2905728.

[66] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," *IEEE Transactions on Services Computing*, vol. 15, pp. 1411–1422, 3 May 2022, ISSN: 1939-1374. DOI: 10.1109/TSC.2020.2993728. [Online]. Available: https://ieeexplore.ieee.org/document/9090992/.

[67] M. Jodayree, M. Abaza, and Q. Tan, "A predictive workload balancing algorithm in cloud services," *Procedia Computer Science*, vol. 159, pp. 902–912, 2019.

[68] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Forecasting cloud application workloads with cloudinsight for predictive resource management," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1848–1863, 2020.

[69] B. Feng, Z. Ding, and C. Jiang, "Fast: A forecasting model with adaptive sliding window and time locality integration for dynamic cloud workloads," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1184–1197, 2023. DOI: 10.1109/TSC.2022.3156619.

[70] I. Baldini, P. Castro, K. Chang, *et al.*, "Serverless computing: Current trends and open problems," *Research advances in cloud computing*, pp. 1–20, 2017. DOI: 10.1007/978-981-10-5026-8_1.

[71] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, Nov. 2019, ISSN: 0001-0782. DOI: 10.1145/3368454. [Online]. Available: https://doi.org/10.1145/3368454.

[72] A. Alhindi, K. Djemame, and F. B. Heravan, "On the power consumption of serverless functions: An evaluation of openfaas," in *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2022, pp. 366–371. DOI: 10.1109/UCC56403.2022.00064.

[73] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 83, pp. 50–59, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.01.022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17316485.

[74] J. Carreira, S. Kohli, R. Bruno, and P. Fonseca, "From warm to hot starts: Leveraging runtimes for the serverless era," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21, Ann Arbor, Michigan: Association for Computing Machinery, 2021, pp. 58–64, ISBN: 9781450384384. DOI: 10.1145/3458336.3465305. [Online]. Available: https://doi.org/10.1145/3458336.3465305.

[75] G. Casale, M. Artač, W. v. d. Heuvel, *et al.*, "Radon: Rational decomposition and orchestration for serverless computing," *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, pp. 77–87, 1 2020. DOI: 10.1007/s00450-019-00413-w.

[76] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and hpc," in *High Performance Computing*, E. Mocskos and S. Nesmachnow, Eds., Cham: Springer International Publishing, 2018, pp. 154–168, ISBN: 978-3-319-73353-1.

[77] L. Schuler, S. Jamil, and N. Kühl, "Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 804–811. DOI: 10.1109/CCGrid51090.2021.00098.

[78] R. Tolosana-Calasanz, G. G. Castañé, J. Á. Bañares, and O. Rana, "Modelling serverless function behaviours," in *Economics of Grids, Clouds, Systems, and Services*, K. Tserpes, J. Altmann, J. Á. Bañares, *et al.*, Eds., Cham: Springer International Publishing, 2021, pp. 109–122, ISBN: 978-3-030-92916-9.

[79] D. Ustiugov, T. Amariucai, and B. Grot, "Analyzing tail latency in serverless clouds with stellar," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, Nov. 2021, pp. 51–62. DOI: 10.1109/IISWC53511.2021.00016.

[80] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, "Benchmarking, analysis, and optimization of serverless function snapshots," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, Virtual, USA: Association for Computing Machinery, 2021, pp. 559–572, ISBN: 9781450383172. DOI: 10.1145/3445814.3446714. [Online]. Available: https://doi.org/10.1145/3445814.3446714.

[81] R. B. Roy, T. Patel, and D. Tiwari, "Icebreaker: Warming serverless functions better with heterogeneity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22, Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 753–767, ISBN: 9781450392051.

DOI: 10.1145/3503222.3507750. [Online]. Available: https://doi.org/10.1145/3503222.3507750.

[82] P. Raith, "Faas-sim: A trace-driven simulation framework for serverless edge computing platforms," *Software Practice and Experience*, 2023. DOI: 10.1002/spe.3277.

[83] S. Arbat, V. Jayakumar, J. Lee, W. Wang, and I. Kim, "Wasserstein adversarial transformer for cloud workload prediction," *Proceedings of the Aaai Conference on Artificial Intelligence*, vol. 36, pp. 12 433–12 439, 11 2022. DOI: 10.1609/aaai.v36i11.21509.

[84] S. Lee, D. Yoon, S. Yeo, and S. Oh, "Mitigating cold start problem in serverless computing with function fusion," *Sensors*, vol. 21, no. 24, 2021, ISSN: 1424-8220. DOI: 10.3390/s21248416. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8416.

[85] N. Mahmoudi and H. Khazaei, "Mlproxy: Sla-aware reverse proxy for machine learning inference serving on serverless computing platforms," *arXiv preprint arXiv:2202.11243*, 2022. DOI: 10.48550/arxiv.2202.11243.

[86] X. Wei, F. Lu, T. Wang, *et al.*, "No provisioned concurrency: Fast RDMA-codesigned remote fork for serverless computing," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, Boston, MA: USENIX Association, Jul. 2023, pp. 497–517, ISBN: 978-1-939133-34-2. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/wei-rdma.

[87] E. van Eyk, J. Scheuner, S. Eismann, C. L. Abad, and A. Iosup, "Beyond microbenchmarks: The spec-rg vision for a comprehensive serverless benchmark," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20, Edmonton AB, Canada: Association for Computing Machinery, 2020, pp. 26–31, ISBN: 9781450371094. DOI: 10.1145/3375555.3384381. [Online]. Available: https://doi.org/10.1145/3375555.3384381.

[88] M. Steinbach, A. Jindal, M. Chadha, M. Gerndt, and S. Benedict, "Tppfaas: Modeling serverless functions invocations via temporal point processes," *Ieee Access*, vol. 10, pp. 9059–9084, 2022. DOI: 10.1109/access.2022.3144078.

[89] Z. liu and X. Xu, "Studying the impact of health education on student knowledge and behavior through big data and cloud computing," *Scientific Programming*, vol. 2022, pp. 1–11, 2022. DOI: 10.1155/2022/4160821.

[90] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Computing Surveys*, vol. 48, pp. 1–43, 3 Feb. 2016, ISSN: 0360-0300. DOI: 10.1145/2856127. [Online]. Available: https://dl.acm.org/doi/10.1145/2856127.

[91] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a" kneedle" in a haystack: Detecting knee points in system behavior," in *2011 31st international conference on distributed computing systems workshops*, IEEE, Minneapolis, MN, USA: IEEE, 2011, pp. 166–171.

[92] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, C Nov. 1987, ISSN: 03770427. DOI: 10.1016/0377-0427(87)90125-7. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0377042787901257.

[93] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[94] I. M. K. Karo, K. MaulanaAdhinugraha, and A. F. Huda, "A cluster validity for spatial clustering based on davies bouldin index and polygon dissimilarity function," in *2017 Second International Conference on Informatics and Computing (ICIC)*, IEEE, Jayapura, Indonesia: IEEE, 2017, pp. 1–6.

[95] C. Kotas, T. Naughton, and N. Imam, "A comparison of amazon web services and microsoft azure cloud platforms for high performance computing," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2018, pp. 1–4.

[96] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17, Shanghai, China: Association for Computing Machinery, 2017,

pp. 153–167, ISBN: 9781450350853. DOI: 10.1145/3132747.3132772. [Online]. Available: https://doi.org/10.1145/3132747.3132772.

[97] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10, Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178, ISBN: 9781605587998. DOI: 10.1145/1772690.1772862. [Online]. Available: https://doi.org/10.1145/1772690.1772862.

[98] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview, ii," *WIREs Data Mining and Knowledge Discovery*, vol. 7, no. 6, e1219, 2017. DOI: https://doi.org/10.1002/widm.1219.

[99] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002. DOI: 10.1109/34.1000236.

[100] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Cloud workload categorization using various data preprocessing and clustering techniques," in *2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23)*, IEEE/ACM, 2023.

[101] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Incorporating data preparation and clustering techniques for workload segmentation in large-scale cloud data centers," in *2023 Fourth International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, IEEE, 2023, pp. 7–14.

[102] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–43, 2016.

[103] S. Vega-Pons and J. Ruiz-Shulcloper, "A survey of clustering ensemble algorithms," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 25, no. 03, pp. 337–372, 2011.

[104] M. Ali, *Pycaret: An open source, low-code machine learning library in python*, PyCaret version 1.0, Apr. 2020. [Online]. Available: https://www.pycaret.org.

[105] W. D. McGinnis, C. Siu, S. Andre, and H. Huang, "Category encoders: A scikit-learn-contrib package of transformers for encoding categorical data," *Journal of Open Source Software*, vol. 3, no. 21, p. 501, 2018.

[106] B. Bengfort and R. Bilbro, "Yellowbrick: Visualizing the Scikit-Learn Model Selection Process," *The Journal of Open Source Software*, 1075th ser., vol. 4, no. 35, Mar. 24, 2019. DOI: 10.21105/joss.01075.

[107] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "An ensemble clustering approach for modeling hidden categorization perspectives for cloud workloads," *Cluster Computing*, pp. 1–25, 2023. DOI: 10.1007/s10586-023-04205-5.

[108] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, IEEE, 2008, pp. 413–422.

[109] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, e1249, 2018.

[110] L. Breiman, "Stacked regressions," *Machine learning*, vol. 24, pp. 49–64, 1996.

[111] K. An and J. Meng, "Voting-averaged combination method for regressor ensemble," in *International Conference on Intelligent Computing*, Springer, 2010, pp. 540–546.

[112] M. L. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. DOI: 10.21105/joss.03021. [Online]. Available: https://doi.org/10.21105/joss.03021.

[113] A. Botchkarev, "A new typology design of performance metrics to measure errors in machine learning regression algorithms," *Interdisciplinary Journal of Information, Knowledge, and Management*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202770279.

[114] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[115] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "A multilevel learning model for predicting cpu utilization in cloud data centers," in *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, 2023, pp. 1016–1023.

[116] M. Daraghmeh, S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman, "Regression-based dynamic provisioning and monitoring for responsive resources in cloud infrastructure networks," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–7. DOI: 10.1109/AICCSA.2018.8612806.

[117] M. Daraghmeh, S. Bani Melhem, A. Agarwal, N. Goel, and M. Zaman, "Linear and logistic regression based monitoring for resource management in cloud networks," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2018, pp. 259–266. DOI: 10.1109/FiCloud.2018.00045.

[118] M. Daraghmeh, I. Al Ridhawi, M. Aloqaily, Y. Jararweh, and A. Agarwal, "A power management approach to reduce energy consumption for edge computing servers," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, pp. 259–264. DOI: 10.1109/FMEC.2019.8795328.

[119] M. Daraghmeh, A. Agarwal, N. Goel, and J. Kozlowskif, "Local regression based box-cox transformations for resource management in cloud networks," in *2019 Sixth International Conference on Software Defined Systems (SDS)*, 2019, pp. 229–235. DOI: 10.1109/SDS.2019.8768643.

[120] M. Daraghmeh, A. Agarwal, R. Manzano, and M. Zaman, "Time series forecasting using facebook prophet for cloud resource management," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, Montreal, QC, Canada: IEEE, 2021, pp. 1–6. DOI: 10.1109/ICCWorkshops50388.2021.9473607.

[121] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Ensemble learning for predicting task connectivity over time in cloud data centers," in *IEEE INFOCOM 2023 - IEEE Conference on*

Computer Communications Workshops (INFOCOM WKSHPS), 2023, pp. 1–6. DOI: 10.1109/INFOCOMWKSHPS57453.2023.10225964.

[122] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Regression-based approach for proactive predictive modeling of efficient cloud cost estimation," in *2023 Tenth International Conference on Software Defined Systems (SDS)*, IEEE, 2023, pp. 65–72.

[123] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[124] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[125] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems: First International Workshop, MCS 2000 Cagliari, Italy, June 21–23, 2000 Proceedings 1*, Springer, 2000, pp. 1–15.

[126] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, vol. 17, pp. 168–192, 1 Jan. 2021, ISSN: 2634-1964. DOI: 10.1016/j.aci.2018.08.003. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1016/j.aci.2018.08.003.

[127] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html.

[128] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Leveraging imbalance and ensemble learning methods for improved load prediction in cloud computing systems," in *2023 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2023.

[129] B. Wang, A. Ali-Eldin, and P. Shenoy, "Lass: Running latency sensitive serverless computations at the edge," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '21, Virtual Event, Sweden: Association

for Computing Machinery, 2021, pp. 239–251, ISBN: 9781450382175. DOI: 10.1145/3431379.3460646. [Online]. Available: https://doi.org/10.1145/3431379.3460646.

[130] S. McAleese, J. C. McLaughlin, F. Detyna, A. Murashev, M. Yilmaz, and P. M. Clarke, "Serverless software engineering – and how to get there.," in *Systems, Software and Services Process Improvement*, M. Yilmaz, P. Clarke, R. Messnarz, and B. Wöran, Eds., Cham: Springer International Publishing, 2022, pp. 75–90, ISBN: 978-3-031-15559-8.

[131] I. Müller, R. Marroquín, and G. Alonso, "Lambada: Interactive data analytics on cold data using serverless cloud infrastructure," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20, Portland, OR, USA: Association for Computing Machinery, 2020, pp. 115–130, ISBN: 9781450367356. DOI: 10.1145/3318464.3389758. [Online]. Available: https://doi.org/10.1145/3318464.3389758.

[132] Y. Yuan, X. Shi, Z. Lei, X. Wang, and X. Zhao, "Smpi: Scalable serverless mpi computing," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, Nov. 2022, pp. 275–282. DOI: 10.1109/IPCCC55026.2022.9894339.

[133] A. Luckow and S. Jha, "Performance characterization and modeling of serverless and hpc streaming applications," in *2019 IEEE International Conference on Big Data (Big Data)*, Dec. 2019, pp. 5688–5696. DOI: 10.1109/BigData47090.2019.9006530.

[134] B. L. Dalmazo, J. P. Vilela, and M. Curado, "Online traffic prediction in the cloud: A dynamic window approach," in *2014 International Conference on Future Internet of Things and Cloud*, IEEE, 2014, pp. 9–14. DOI: 10.1109/ficloud.2014.12.

[135] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *Acm Computing Surveys*, vol. 54, pp. 1–36, 11s 2022. DOI: 10.1145/3510412.

[136] D. Senthil and G. Suseendran, "Efficient time series data classification using sliding window technique based improved association rule mining with enhanced support vector machine,"

International Journal of Engineering & Technology, vol. 7, no. 3.3, p. 218, 2018. DOI: 10.14419/ijet.v7i2.33.13890.

[137] M. Shahrad, R. Fonseca, I. Goiri, *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, Jul. 2020, pp. 205–218, ISBN: 978-1-939133-14-4. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad.

[138] M. Daraghmeh, A. Agarwal, and Y. Jararweh, "Optimizing serverless computing: A comparative analysis of multi-output regression models for predictive function invocations," *Simulation Modelling Practice and Theory (In Review, Submitted in December 2023)*, 2024.