Computing and Informatics, Vol. 43, 2024, 1053–1083, doi: 10.31577/cai_2024_5_1053

MULTILEVEL ENSEMBLE MODEL FOR LOAD PREDICTION ON HOSTS IN FOG COMPUTING ENVIRONMENT

Shabnam Bawa, Prashant Singh Rana, Rajkumar Tekchandani

Department of Computer Science and Engineering
Thapar Institute of Engineering and Technology
Patiala, India
e-mail: {sbawa_phd18, prashant.singh, rajkumar.tekchandani}@thapar.edu

Abstract. With the growing demand for various IoT applications, fog nodes frequently become overloaded. Fog computing requires effective load balancing to maximize resource utilization. It is essential to determine the load on host to obtain workload consolidation. Various random parameters, including CPU utilization, the number of CPU cores, RAM, memory allocated, memory available, disk I/O, and network I/O are employed to better comprehend host workloads. In the proposed work, the host's load status is detected using an ensemble approach into three categories: under-loaded, balanced and overloaded. Further, the proposed work considers three case studies and varying numbers of virtual machines (VMs) are executed with various parameter combinations. In each case study, a different number of VMs are executed in parallel on two different platforms. In the proposed study, we predicted the load on multiple hosts by employing a variety of advanced machine-learning models. To construct an ensemble model, we selected models with higher accuracy based on retrieved performance evaluation criteria. The ensemble method is applied to deal with the worst-case scenario of the model prediction. For a selected number of case studies, the Random Forest model, Ada Boost Classifier, Gradient Boost and Decision Tree models perform better than other models. These state-of-the-art predictive models are outperformed by our proposed ensemble model and achieves an improved accuracy of nearly 82 % by correctly classifying hosts as overloaded, underloaded and balanced.

Keywords: IoT, virtual machine, containers, fog computing, machine learning, load prediction

1 INTRODUCTION

Mobile Internet, Cyber Physical Systems, along with the Internet of Things (IoT) have empowered a multitude of entities, including individuals, machinery and objects, to establish continuous connections with the digital realm, irrespective of time or location. Data is being generated in unprecedented quantities and variety [1, 2]. Cisco predicts that by 2023, the number of networked gadgets per user will increase from 2.4 in 2018 to 3.6. This surge will drive the total count of networked devices from 18.4 billion in 2018 to an estimated 29.3 billion by 2023. This proliferation in connected devices corresponds to a remarkable surge in data generation rates. In a recent study including a healthcare related IoT application, it has been found that thirty million people produce 25 000 data tuples per second collectively [3]. Given this tremendous data volume, the current processing and storage capacities fall short of the demands. Conventional computing methods, including distributed computing and cloud computing, struggle to cope with this data deluge. As a more pragmatic solution, fog computing has emerged, bridging the gap between network edge devices and centralized cloud centers to efficiently address these limitations [4]. Fog Computing is a middle man between the Internet of Things and the cloud that brings information closer to the sensors [5]. Nevertheless, considering the increasing demand for a variety of IoT applications, fog nodes frequently become overloaded, degrading the response times of IoT apps with latency constraints and as a result, compromising users' quality of experience [6]. Understanding workload characteristics and underlying data centers are critical for both data center operators and fog service providers to sustain the adoption of fog data centers and increase data center operators' ability to tune existing and build new resource management approaches. The architecture of fog computing is shown in Figure 1.

Google Data Centers reportedly utilized an estimated 260 million Watts of electrical power in 2013, accounting for one percent of global energy consumption [7]. A 2020 New York Times article states that data centers consumed around 1 % of global electrical output 2018. The inefficient use of hardware resources is the cause of high energy consumption. Servers inactive can use up to 60 % of their peak power [8, 9].

Virtualization capabilities, which were initially introduced by IBM in the early 1960s as a transparent method to facilitate continuous and direct access to powerful computers by numerous users, represent a valuable method for addressing energy inefficiencies [10]. This virtualization technique abstracts the physical resources that underlie a system, thereby generating an integrated view of a resource pooling within fog data centers. where a single server can accommodate multiple virtual machines. Virtualization provides several advantages for distributed systems, including increased resource utilization, enhanced isolation, improved manageability, and heightened reliability [11]. It becomes crucial to assess the host's workload to achieve effective workload consolidation. This is useful when migrating containers from a particular virtual machine to another in order to reduce the number of service level agreement (SLA) violations and balancing of loads violations.

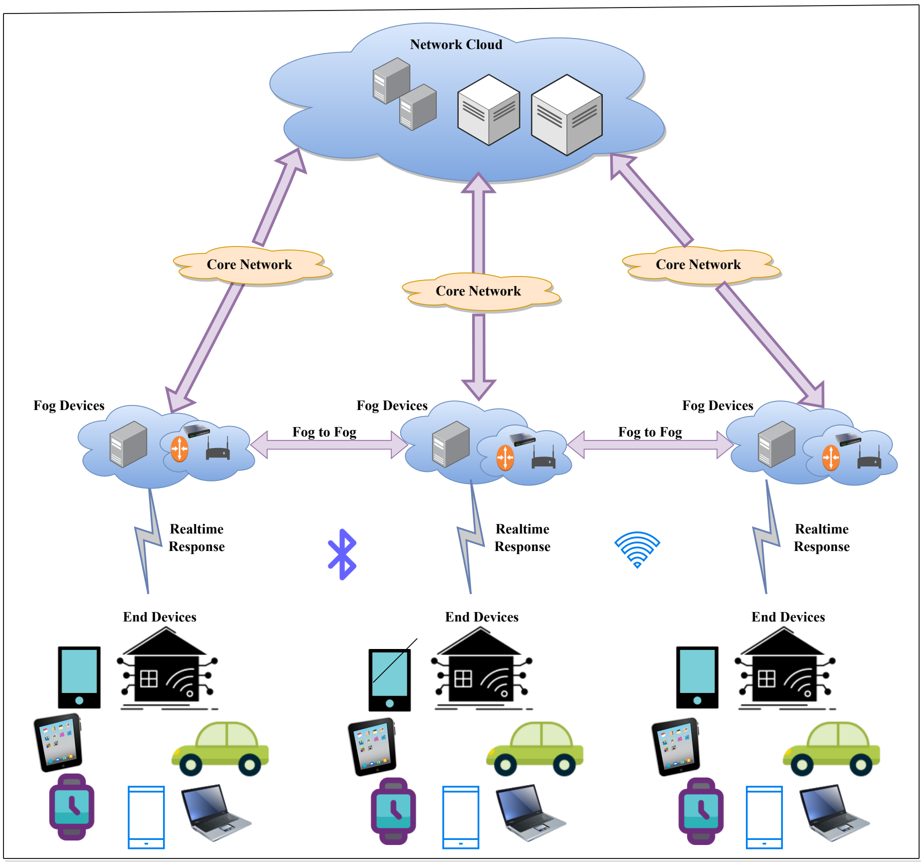

Figure 1. The architecture of fog computing

A novel framework has been developed to classify hosts into overloaded, underloaded and balanced hosts. Then the process of workload consolidation in a fog environment is discussed by detecting overloaded hosts. Containers represent a groundbreaking innovation in the era of cloud computing due to their lightweight nature, simplified configuration and administration, and substantial reduction in startup times [12]. Consequently, a diverse range of applications can be seamlessly executed within these containers. To achieve this, we have employed Podman to run containerized applications on VMs. We can run heavy applications through containers as these are lightweight. The two platforms we have used are shown in Table 6. We can efficiently run heavy applications through containers, such as applications related to heavily loaded images due to high resolution. Containers are executed on the virtual machines, created using the command line with various random parameters, including the CPU, number of cores, RAM, memory allocated, memory available, disk Input/Output and network I/O in order to understand the host workloads better. Figure 2 shows the container-based virtual machines.

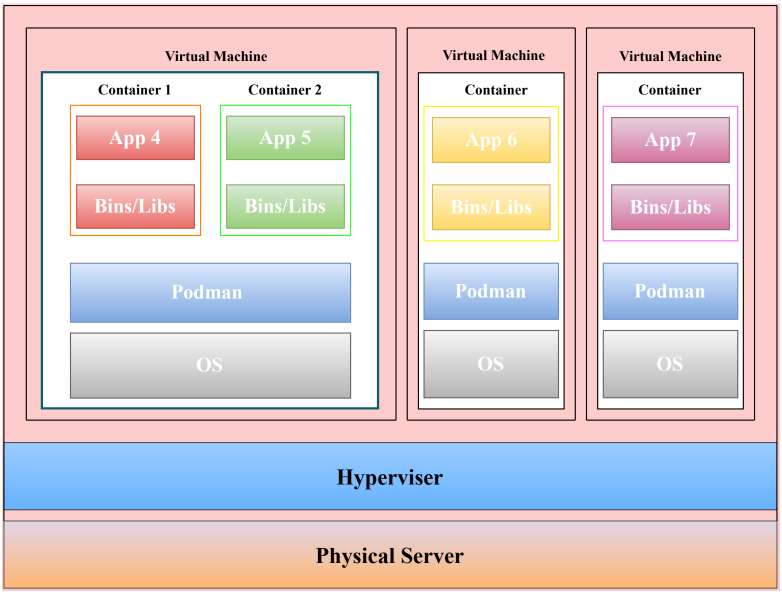

Figure 2. Container-based virtual machines

In this work, own dataset is generated using various types of VMs. The entire dataset is merged, after which machine learning models are applied to detect loads on multiple hosts. This paper consists of various sections as follows: Section 2 describes the related work and significant contributions. Section 3 describes materials and methods and gives an overview of the various parameters used for dataset generation, dataset generation using modeling and simulation, including the transformation of data, Load prediction on Physical Machine (PM), Service Level Agreement Metric and various machine learning methods used to predict the load on multiple hosts. Section 4 includes the methodology, multilevel ensemble model proposed, and case studies. Section 5 discusses model evaluation. The results analysis, comparison, and discussion are discussed in Section 6. Section 7 contains the conclusion and the next steps.

2 RELATED WORK

Several studies have examined the prediction of load for cloud hosts and implemented various methods to address the problem of workload prediction, thereby enhancing the accuracy of models. Present approaches for workload prediction primarily improve model accuracy by employing techniques such as regression, statistics, machine learning and RNN-based methods. In cloud computing, workload prediction based machine learning techniques include parametric, non parametric, neural network and deep learning approaches. Parametric models use learning methods like the kalman filter [13] and the automatic regression moving average model (ARIMA) [14]. Support vector regression (SVR) and neural network (NN)-based methodologies are non parametric approaches.

For prediction of load on host, different time series-based approaches are suggested, such as ARIMA [15], linear regression [16], exponential averaging [17] and others. Calheiros et al. [14] proposed an ARIMA-based load prediction model for cloud computing in order to accomplish predictive scaling for virtual machine instances. Tan et al. [18], proposed an aggregation method for prediction of traffic flow that incorporates ARIMA [14], moving average, exponential smoothing, and neural networks. The utilization of time series prediction methods is sub optimal in non linear scenarios due to the non linearity and linearity of the time series produced by varying resource loads. A large number of machine learning techniques, including recurrent neural networks (RNN), support vector machines (SVM) and artificial neural networks (ANN), are also employed for load prediction. Janardhanan et al. [19] used Long Short Term Memory (LSTM) to forecast CPU utilization. LSTM exhibits superior performance to time series, according to some results. Fan et al. [20] enhanced the performance of the Temporal Convolutional Network (TCN) across various time series prediction tasks. Kumar et al. [21] proposed a self-directed workload prediction method to capture prediction error trends by calculating deviations. By optimizing solutions for the prediction error feedback window, cluster size, and learning rate, its performance can be enhanced further. Khan et al. [22] conducted a comprehensive study of numerous machine-learning algorithms, including linear regression, Regression with Ridges, ARD Regression, Elastic Net, along with deep learning techniques like the Gated Recurrent Unit. Patel and Misra [23] presented a workload forecast model utilizing neural networks and a self-adapting evolutionary algorithm. Several neural network-based methods for improving prediction accuracy have been proposed. However, accuracy of these methods is insufficient and considers specific workload.

Table 1 provides a summary of the comparative analysis of various approaches for load prediction in diverse computing paradigms. As per Table 1, load prediction is evaluated on cloud computing platform whereas our approach performs load prediction on nodes in fog computing environment. As cloud environment generate enormous amount of data volumes, so conventional computing methodologies, like distributed computing and cloud computing are unable to handle challenges such as reducing latency, response time and consumption of energy. As shown in Figure 1, fog computing effectively bridges the gap between network gadgets at the network's periphery and centrally located cloud computing facilities, emerges as a simpler approach to manage these challenges. In our proposed work, a multilevel ensemble model for classifying hosts as overloaded, underloaded, or balanced is presented. Multiple features are extracted in the proposed work. All of the features utilized by the proposed model, including CPU utilization, number of cores, RAM, allocated memory, disk I/O, available memory and network I/O, are extracted from running hosts. The experimental outcomes demonstrate that the performance of the proposed multilevel ensemble model for prediction of load gives more accuracy as compared to existing state-of-the-art approaches, as discussed in Table 1.

2.1 Major Contribution

This paper's contributions are summed up as follows:

1. A multilevel ensemble model has been developed to classify hosts. This proposed approach uses various parameters like CPU usage, number of cores, RAM, memory allocated, disk I/O, memory available and network I/O. The load forecasting of hosts based on these parameters has been completed utilizing modern artificial intelligence algorithms.
2. The dataset is generated by executing a different number of VMs in parallel on two platforms with different configurations.
3. In our proposed method for categorizing hosts as overloaded, underloaded, or balanced, we employed a range of supervised machine learning models. These models encompassed Random Forest, Light Gradient Boosting Machine, K-Neighbors Classifier, Ridge Classifier, Linear Discriminant Analysis, Decision Tree, Gradient Boost Classifier, Naive Bayes Classifier, Ada Boost Classifier and Extra Tree Classifier. The ensemble technique is used to deal with the worst-case scenario of the model prediction.

3 MATERIALS AND METHODS

3.1 Parameters Used for DataSet Generation

The dataset contains ten thousand records with eleven performance parameter values. For predicting whether a host is overloaded, underloaded, or balanced, the total CPU usage parameter is retained as the target variable, while the remaining variables in the dataset serve as input for machine learning models. Each row of the dataset represents a performance metric observation. Each row is formatted as follows:

CPU cores: The quantity of number of virtual CPU cores allocated.

User: The percentage of CPU consumption that happened while the user was executing (application).

nice: Percentage of CPU utilization during user-level execution with pleasant priority.

Total Usage of CPU: in terms of percentage.

Ideal CPU: The percentage of time the CPU or CPUs were idle when no disk I/O requests were pending.

Paper	**Method**	**Computing Paradigm**	**Datasets**	**Metrics**	**Work Done**
Calheiros et al. [14]	ARIMA Model	Cloud Computing	Real traces of user requests to web servers.	CPU Utilization	ARIMA model was used to predict future workload using actual Wikimedia Foundation web server request traces and evaluate its accuracy.
Gao et al. [24]	Machine Learning	Cloud Computing	Google cluster trace	CPU	A clustering-based workload prediction method is designed.
Kumar and Singh [25]	Artificial neural network	Cloud Computing Environment	NASA HTTP traces and Saskatchewan HTTP traces	SLA Violations	A paradigm for workload forecasting is presented that utilizes an artificial neural network and an algorithm for capable of self-adapt differential evolution.
Ramezani [26]	fuzzy workload prediction Algorithm	Cloud Computing Environment	Historical workload data- two datastores	VM CPU Usage	Developed a fuzzy work estimation method that monitors both previous and current VM's CPU consumption and forecasts physical machine utilization.
Yang et al. [27]	linear regression model	Cloud Computing	N/A	CPU Usage, SLA Violation	Utilizing a linear regression approach to forecast the workload, an automated scaling mechanism for scaling virtual assets in service clouds is proposed.
Qiu et al. [28]	Deep Belief Networks	Cloud Computing Environment	PlanetLab	CPU Utilization	A load forecasting method based on deep machine learning is proposed. Improved the predictability of the CPU utilization prediction when compared to current methods.
Jheng et al. [29]	Grey Forecasting model	Cloud Computing	four historical workloads of PMs and VMs	Power Consumption	In data centers, the grey model is used to develop a method for workload prediction and to reduce the electrical power consumption of PMs.
Yu et al. [30]	multilayer perceptron neural network	Cloud Environment	Google Dataset	CPU Usage	An approach to workload forecast in the cloud that estimates the workload patterns of new tasks based on the statistical properties of a set of tasks.
Patel and Misra [23]	Deep learning Models	Cloud Computing	real workload traces of PlanetLab	CPU Usage	Predicts large-scale, highly data-driven systems for distributed decision making, such as hot spot mitigation, cold spot mitigation, benchmark violation and service level agreements violation.
Eramo et al. [31]	Convolutional Short-Term Memory neural network	Cloud Computing	N/A	QoS degradation cost	Using a straightforward VNFI reconfiguration and placement algorithm, the effectiveness of a LSTM neural network system for resource prediction and allocation is evaluated.
Caron et al. [32]	Pattern Matching	Cloud and Grid Computing	Traces of Animoto, LCG, Nordugrid and SHARCNET	Min prediction error, Max prediction error, Avg prediction error	It identify previous instances resembling the current short-term work history.
Roy et al. [33]	A model-forecasting algorithm	Cloud Computing	World Cup Soccer 1998 Workload	SLA Violation, reconfiguration cost	For resource automated scaling, a model-predictiv e method for workload forecasting has been devised and implemented.
Janardhanan et al. [19]	Time series forecasting	Cloud Computing	Google's cluster dataset	CPU Usage	The utilization of LSTM Networks to predict the CPU utilization of machines in data centres has demonstrated a substantial advancement in comparison to the ARIMA models.
Kumar et al. [21]	self directed workload forecasting method (SDWF)	Cloud Computing	six different real world data traces	quality of experience (QoE)	A self-directed workload forecasting method is proposed to enhance prediction quality and enhance cloud system service quality.
Proposed Work	Ensemble Model	Fog Computing Environment	Generated Real workload data	CPU Usage, Power Consumption	Developed a multilevel ensemble model to classify hosts based on workload prediction.

Table 1. Comparative analysis of existing approaches of load prediction

Memory Total: the memory given to the system in terms of KB.

Memory Free: the memory that is actively not used in terms of KB.

Memory Available: An estimate of the amount of memory available to execute new applications without swapping in additional KB.

Disk Read/Write Throughput: in terms of KB/s.

Network Received Throughput: Received network throughput in terms of MBITS/s.

Network Transmitted Throughput: Transmitted network throughput in terms of MBITS/s.

Basic features statistics are calculated for the statistical characterization, including the min, the standard deviation (SDev), the mean and the unitless coefficient of variation (CoV). Table 2 shows basic statistics of various features used in this work. Table 3 demonstrates the sample of the dataset used in the course of this work.

	User	**Nice**	**Idle**	**Total CPU Usage**	**Network Transmitted Throughput**	**Network Recieved Throughput**	**CPU Cores**	**MemTotal**	**MemFree**	**MemAvailable**	**Disk Write Throughput**
CoV	60.84	126.38	23.39	63.81	226.72	223.17	44.96	48.03	103.84	86.15	457.92
mean	12.58	0.36	73.18	26.82	131.18	137.06	6.01	3 690 608.00	264 915.80	1 318 698.00	0.18
std	7.65	0.45	17.11	17.11	297.42	305.87	2.70	1 772 700.00	275 097.20	1 136 045.00	0.81
min	1.01	0.03	27.96	2.35	0.00	0.00	1.00	1 026 028.00	53 468.00	21 752.00	0.00
25%	6.43	0.12	62.16	14.09	14.10	14.00	4.00	1 537 580.00	101 126.00	446 484.00	0.00
50%	11.59	0.20	81.07	18.93	19.10	18.80	7.00	4 100 348.00	139 480.00	1 127 272.00	0.02
75%	17.13	0.29	85.91	37.85	23.45	23.20	9.00	4 666 636.00	298 198.00	1 616 788.00	0.09

Table 2. Basic statistics of various features

3.2 Dataset Generation Using Modeling and Simulation

To better understand the workloads of hosts, virtual machines are generated through the command line through different random parameters. The proposed model targets a Container as a Service (CaaS) environment where an application is executed on containers. Various applications are running on the containers and containers are executed on these virtual machines. A whole dataset is merged, a new dataset is generated and then a set of machine learning models is applied to them to detect loads on various hosts. The flow chart for creating VMs through the command line is shown in Figure 3. Table 4 below describes performance parameters and their range used for the construction of VMs through the command line for this work.

User	**Nice**	**System**	**IOWait**	**Idle**	**CPU Usage**	**Network Transmitted Throughput**	**Network Recieved Throughput**	**CPU Cores**	**Mem Free**	**Memory Available**	**Disk Write Throughput**
16.05	0.1	21.49	13.78	48.59	51.41	17.0 mbits/sec	15.3 mbits/sec	9	54 756	37 168	1.06206131
16.11	0.1	21.85	13.92	48.02	51.98	20.4 mbits/sec	16.9 mbits/sec	9	61 428	47 468	0.985242367
15.96	0.1	22.47	14.21	47.27	52.73	12.7 mbits/sec	10.3 mbits/sec	9	54 820	21 752	0.130439281
15.82	0.1	23.05	14.45	46.58	53.42	11.3 mbits/sec	11.2 mbits/sec	9	55 792	41 660	0.442959309
15.58	0.09	24	14.65	45.68	54.32	16.3 mbits/sec	16.8 mbits/sec	9	53 468	26 824	10.23108006
19.2	0.32	15.92	1.76	62.79	37.21	25.1 mbits/sec	20.0 mbits/sec	4	514 724	1 415 904	0.375194073
20.16	0.32	16.4	2.38	60.73	39.27	23.1 mbits/sec	24.7 mbits/sec	4	619 868	1 403 036	0.002605438
22.34	0.3	18.25	2.87	56.24	43.76	23.4 mbits/sec	19.8 mbits/sec	4	188 060	970 992	0.495999813
22.6	0.3	18.16	4.62	54.32	45.68	18.8 mbits/sec	21.4 mbits/sec	4	223 156	1 319 912	0.70169425
29.94	0.23	25.14	6.64	38.04	61.96	22.7 mbits/sec	11.9 mbits/sec	4	708 996	1 570 532	0.004311085
30.44	0.23	25.59	6.5	37.24	62.76	13.7 mbits/sec	19.4 mbits/sec	4	891 736	1 558 736	0.625935555

Table 3. Sample dataset

SN	**Parameter**	**Range**
1	CPU cores	4–10
2	RAM	1–6 GB
3	Disk Write Throughput (file size)	1–5 GB
4	Disk Size	80 GB
5	Memory	2 048–8 192 MB

Table 4. Range of various parameters used for VM creation

3.2.1 Transformation of Data

Static thresholds T_{ol} and T_{ul} are used to decide if a host's load is overloaded, underloaded and balanced, as shown in Equation (1), respectively.

$$\text{Host Status} = \begin{cases} \text{Overloaded}, & \text{if } U_{(i,t)} > T_{ol}, \\ \text{Underloaded}, & \text{if } U_{(i,t)} > T_{ul}, \\ \text{Balanced}, & \text{if } T_{ul} < U_{(i,t)} > T_{ol}, \end{cases} \tag{1}$$

where T_{ol} and T_{ul} represent the overloaded and underloaded thresholds, respectively, and $U_{(i,t)}$ is the utilization of CPU of host i at a particular time t. The power consumption of the CPU is considered to be a critical factor that exhibits the greatest variability in power usage based on its utilization rate. CPU Utilization is considered a target variable, and the CPU Utilization feature binning takes place. The transformation of the dataset is done to label the data and classifies them into three classes: overloaded (O), underloaded (U) and balanced (B). The sample of dataset transformation is shown in Table 5.

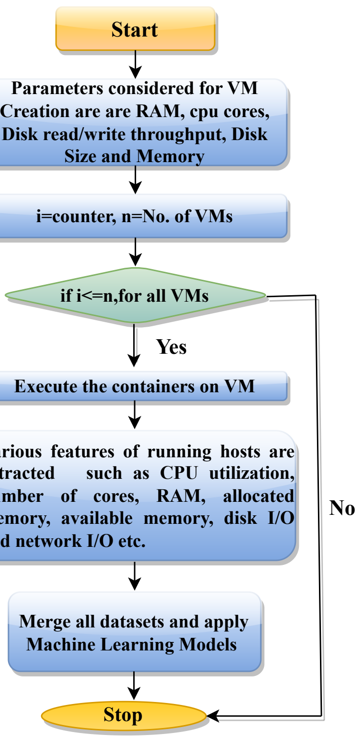
Start
Parameters considered for VM Creation are are RAM, cpu cores, Disk read/write throughput, Disk Size and Memory
i=counter, n=No. of VMs
if i<=n,for all VMs
Yes
No
Execute the containers on VM
Various features of running hosts are extracted such as CPU utilization, number of cores, RAM, allocated memory, available memory, disk I/O and network I/O etc.
Merge all datasets and apply Machine Learning Models
Stop

Figure 3. Flow chart of data generation

User	Nice	Idle	Total CPU Usage	Network Transmitted Throughput	Network Recieved Throughput	CPU Cores	Mem Free	Memory Available	Disk Write Throughput	Label
19.63	1.22	74.76	25.24	9.07	4.77	1	1 430 276	5 860 808	0.668997049	U
57.72	18.16	0.05	99.95	31.4	27.6	2	1 048 228	2 578 964	0.04445672	O
56.19	20.9	0.04	99.96	28.3	27.8	2	676 580	2 209 468	0.374528646	O
25.18	0.97	68.5	31.5	43.5	46.4	3	546 796	3 116 432	0.005527258	B
19.93	1.2	74.36	25.64	5.23	4.93	4	1 406 644	5 900 648	0.769234896	U
24.93	0.97	68.79	31.21	50.6	52.7	3	636 200	3 210 656	1.203319788	B
24.85	0.97	68.89	31.11	45	49.7	3	674 404	3 250 004	1.265292645	B
57.29	19.26	0.05	99.95	27.2	27.4	2	906 184	2 437 896	0.002002001	O

Table 5. Sample of dataset transformation

3.2.2 Load Prediction on Physical Machine

A virtual machine (VM) behaves like a physical machine (PM). So each VM has its CPU, memory and bandwidth. The key component for estimating the energy consumption of hosts is the CPU's power consumption. CPU utilization presents the most significant variance in power consumption concerning its utilization rate. CPU utilization is regarded as a target variable. Load on the i^{th} VM can be calculated using Equation (2) as follows:

$$VM_{load}^{i} = VM_{CPU}^{i}. \tag{2}$$

Determine the weight on the PM since multiple VMs operate on each host. Thus, as Equation (3) illustrates, the overall load on the PM equals the sum of all the VM loads.

$$\mathrm{PM}_{load}^{m} = \sum_{i=1}^{n} \frac{\mathrm{VM}_{load}^{i}}{n}, \tag{3}$$

where n represents the number of virtual machines on the m^{th} host.

3.2.3 Service Level Agreement Metric

This effectiveness metric represents the mean percentage of service level agreement (SLA) violations. It becomes relevant when virtual machines fail to obtain the requested resources or when the shared host's average computing capacity is not allocated to the requested virtual machines, as discussed in [34]. This metric directly influences the Quality of Service (QoS) level. Equation (4) outlines the calculation for the average SLA violation as follows:

$$A_{SLA} = \frac{\sum_{j=1}^{q} \frac{RM_j - AM_j}{RM_j}}{q}. \tag{4}$$

In this context, A_{SLA} signifies the average unallocated VM Million Instructions Per Second (MIPS), which results in a decline in performance. RM represents the requested MIPS by a VM, AM represents the MIPS actually allocated to the VM during its operation, and q signifies the total number of VMs.

3.3 Feature Importance

The term feature importance refers to the procedure of feature selection from a given dataset that makes the greatest contribution to the classification of the trained dataset and the prediction of the objective variable.

PyCaret is a Python library [35] designed for automating machine learning processes and workflows. PyCaret has the capability to determine the significance of a feature. The Feature Importance operation of PyCaret finds out the features that have the greatest impact on the accuracy of classification in the following sequence.

- Generation of dataset;
- Multi-classification;
- Selection of relevant features.

Following the generation of the dataset, preprocessing is performed, including normalization, transformation, PCA and outlier removal. During data preprocessing, unnecessary columns such as system, iowait and memtotal are removed, as shown in Table 2. Further, the data is classified and as a result, we have determined that the proposed ensemble model performs best with an average accuracy of 82%. The feature importance can be derived from the dataset once the classification result has been obtained. We must begin by defining the optimal model, which we refer to as the proposed ensemble model. The proposed model includes various models including random forest. Random Forest builds a large number of decision trees during training to generate the mode of classes or the average prediction of individual trees. Understanding the value of features of dataset helps to gain insights into the dataset and enhance the model by focusing on the most informative characteristics. It could improve classification problem feature engineering, model understanding and feature selection decisions. Moreover, the model is plotted according to the importance of the features on the basis of information gain theory. From the feature importance plot as shown in Figure 4, we can conclude that with respect to the generated dataset, total usage of CPU is the most important feature, followed by CPU cores and user.

3.4 Machine Learning Methods

Various machine learning models are used to classify load on hosts as overloaded, underloaded and balanced. The details of these models are given below:

- A Random Forest Classifier: It is a method of supervised machine learning that employs decision trees for various applications including classification, regression, and others. Using a randomly chosen training dataset, the RF classifier generates a set of trees of decision. Essentially, it is a collection of decision trees (DT) generated by choose a random portion of the training set.
- Gradient Boosting Classifier: It is a sequential learning ensemble approach in which the model's efficacy increases over time. Using this method, the model is constructed successively. With the addition of each feeble learner, a new model is developed, resulting in a more accurate estimation of the response of the variable.
- Decision Tree Classifier: A decision tree is a graphical representation resembling a tree that models decision-making processes, including prospective outcomes, expenses for inputs, and utility factors. This algorithm is relevant to both continuous and categorical variables as outputs and lies under the category of trained learning methods.

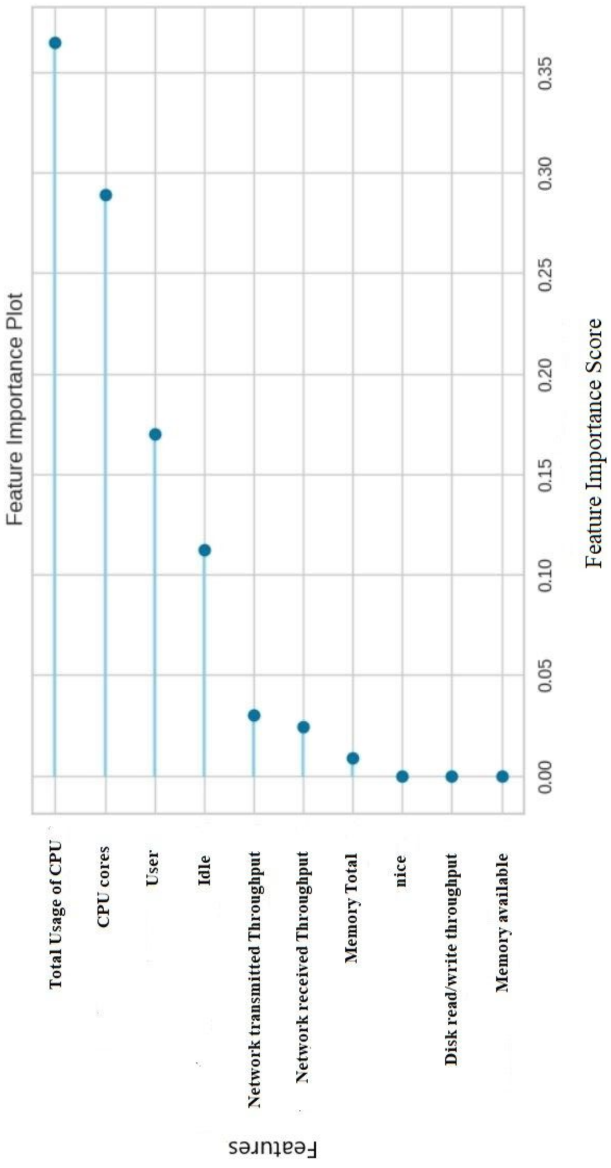

Figure 4. Feature importance for the proposed work

- Extra Tree Classifier: This class implements a meta-estimator that uses averaging to improve predicted accuracy and prevent overfitting by fitting multiple randomized decision trees to different sub-samples of the dataset.

- Ada Boost Classifier: It integrates multiple classifiers to increase classifier precision. AdaBoost generates a robust classifier by combining several underperforming classifiers, resulting in an excessively accurate classifier. Its fundamental principle is to establish the weights of classifiers and teach the data set in each iteration so that accurate predictions can be made for out-of-the-ordinary observations.

- K Neighbors Classifier: The KN Classifier is one of the most fundamental algorithms for classification in the field of machine-learning. Among the applications found in the domain of supervised learning are pattern recognition, data extraction, and intrusion detection. It makes no assumptions regarding data distribution.

- Linear Discriminant Analysis: LDA is a learning with supervision algorithm used for solving classification issues in learning under supervision. It is a method for determining the optimal method for classifying a dataset using a linear combination of features. LDA functions by projecting the data onto a lower-dimensional space that maximizes the separation of various classes. It determines the orientations in the feature space that most effectively separate the various data classes.

- Light Gradient Boosting Machine: Gradient boosting and tree-based learning are utilized by Light GBM. Unique algorithm Light GBM possesses multiple parameters. The dataset is swiftly expanding. Traditional data science methods battle to supply correct results. Light GBM manages large data sets with minimal memory. Based on Ridge regression, the resulting Ridge Classification converts label data to the interval $[-1, 1]$ and employs regression to solve the problem. Using multiclass data and multioutput regression, it selects the target class with the best value of prediction.

- Ridge Classifier: Derived from the Ridge regression approach, the Ridge Classifier transforms label data into the range $[-1, 1]$ and employs regression techniques to address the problem. When dealing with multiclass data, It employs multioutput regression and chooses the target class with the maximum predictive value.

- Naive Bayes Classifier: The naive bayes model is constructed using the principle of Bayes, creating a compilation of classification algorithms. It takes that each pair of classified features has no relationship with the others, a foundational concept shared by every algorithm in this family.

4 METHODOLOGY

The workflow of the present work is shown in Figure 5. The dataset is generated using different numbers of VMs. The extracted feature vector is fed into the various machine-learning models. Further, an ensemble model is proposed and the performance is computed accordingly by predicting the load on the host.

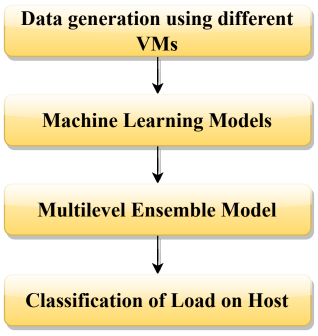
Data generation using different VMs
Machine Learning Models
Multilevel Ensemble Model
Classification of Load on Host

Figure 5. Flowchart of proposed scheme

4.1 Proposed Multilevel Model

The ensemble method is used to manage the model prediction's worst-case scenario. The current study concentrates on the model's false and accurate predictions and a multilevel ensemble model is used to deal with false and accurate predictions. As depicted in Figure 6, a total of seven models, including the RF Classifier, DT Classifier, Linear Discriminant Analysis Classifier, NB, ET Classifier, KN Classifier, and Ridge Classifier, are integrated to enhance accuracy. Seventy percent (70 %) of the dataset is used for training, while the rest of the thirty percent 30 % is reserved for testing. The proposed approach comprises three distinct phases, each of which is explained below:

Phase I: The RF Classifier, DT Classifier, KN Classifier, and NB learned from 70 % of the data set and generate predictions from 30 % of the data.

Phase II: The Linear Discriminant Analysis model is trained using the inaccurate predictions generated by two models, namely the Decision Tree (DT) and K Nearest Neighbor Classifier in Phase I. Ridge Classifier model is trained using the inaccurate predictions generated by two models, namely the RF Classifier and NB, in Phase I.

Phase III: The false predictions from Phase II are combined with the accurate predictions from Phase I. This new combined dataset is used to train the Extra tree classifier model, which makes conclusive predictions. This method refines

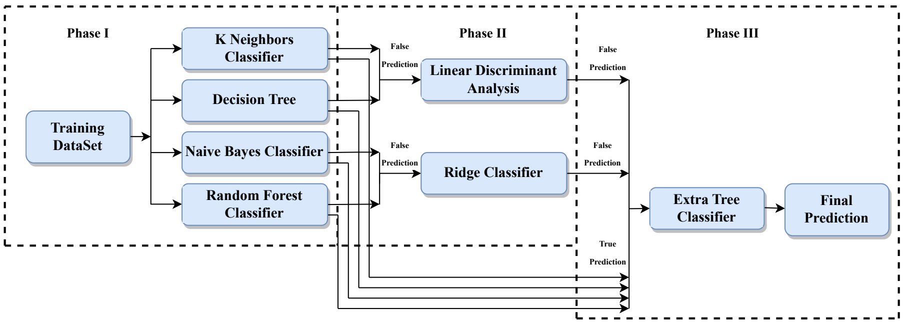

Figure 6. Multilevel ensemble model

accurate and inaccurate predictions to produce a model proposal that is precise. The purpose of incorporating accurate predictions into other models is to combat false-positive results. The data is passed through seven models because, ideally, these models will use the data to produce accurate and reliable results.

4.2 Case Study

Three types of cases are considered for evaluation: In different cases, there are different numbers of virtual machines running in parallel and two types of platforms are used to run these VMs. Oracle VM VirtualBox has been used to run these VMs. Each VM's dataset has 11 parameters: user, nice, system, iowait, CPU Usage, idle, write throughput, memory-free and memory available etc. with CPU usage being the target variable. Table 6 provides an overview of three cases and the platforms used.

Case Study	**Number of VMs Running in Parallel**	**Platform 1**	**Platform 2**	**Software**
Case I	4 VMs (2 on Platform 1 and other 2 on 2nd Platform)	Processor 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80 GHz 2.80 GHz	INTEL(R) XEON(R) CPU E5-2650 V2 @ 2.40 GHZ	Oracle VM VirtualBox, Python (OpenCV)
Case II	6 VMs (3 on Platform 1 and other 3 on 2nd Platform)			
Case III	10 VMs (5 on Platform 1 and other 5 on 2nd Platform)	RAM 12 GB	RAM 16 GB	

Table 6. Different cases considered for evaluation

Case 1: In the first case, four VMs run in parallel, two on the first platform and two on the second. Table 8 shows the detailed comparison of different supervised learning models with ensemble model proposed. The models have been compared in various ways like without pre-processing and applying different pre-processing methods on the generated dataset, like normalization, outlier removal, transformation, PCA and feature selection. When outliers are removed and the dataset is normalized simultaneously, the RF, GBC classifier, AB and DT models perform equally well.

Case 2: In the second case, six VMs run in parallel, three on the first platform and three others on the second. Table 9 shows the detailed comparison of different supervised models with the ensemble model proposed. It has been observed that normalizing the dataset enhances the accuracy of the model. Conversely, when PCA is applied to the dataset, the accuracy of the model decreases.

Case 3: In the third case, ten VMs are running in parallel, five on the first platform and five others on the second platform. Table 10 provides a comprehensive comparison of various artificial intelligence models and the suggested ensemble model. The performance of our proposed model is marginally superior to that of all other models, with a maximum accuracy of around 82 % after the normalization of the dataset.

As previously stated, we have presented three cases. We have observed our proposed model performance through different cases by increasing the number of VMs in each case. As per our results, as shown in Tables 8, 9 and 10, it been seen that increasing the amount of VMs makes our proposed model work better.

5 MODEL EVALUATION

5.1 Evaluation Parameters

Various metrics, including feature importance, recall, precision, the F1 score, Kappa, and accuracy, are employed to assess the model's performance. This study comprehensively evaluates all these parameters. To investigate the robustness of the proposed model, repeated cross-validation with k folds is utilized. Below, we provide explanations for these metrics.

5.1.1 Precision

It assesses the model's ability to correctly predict true classes. Precision is determined by dividing the count of true positive predictions by the total of true positive predictions and fake positives. This precision calculation is based on a specific formula, as shown in Equation (5).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{5}$$

5.1.2 Recall

Recall quantifies the fraction of predicted positive classes. It is determined by dividing the total number of true positive and deceptive negative events by the number of true events that were positive. The formula for calculating the recall is shown in Equation (6).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{6}$$

5.1.3 F1-Score

It is a metric that represents the weighted average of recall and precision scores. It is a value that ranges from 0, indicating the poorest performance, to 1, representing the highest performance. The F1-Score is a statistical measure that combines both precision and recall to offer a complete assessment. Equation (7) is used to figure out F1-Score:

$$\text{F1-Score} = \frac{2}{(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}})}. \tag{7}$$

5.1.4 Cohen's Kappa (Kappa)

It determines how accurate the model's predictions are. The stronger the model, the lower the Kappa value. The Kappa statistic developed by Cohen is ideal for addressing multi-class and unbalanced class problems. Cohen's Kappa is calculated using Equation (8).

$$\text{Kappa} = \frac{p_o - p_e}{1 - p_e}. \tag{8}$$

where p_o represents the observed agreement and p_e represents the anticipated agreement. It indicates by how much our classifier outperforms a classifier that makes random predictions based on the frequency of each class. Cohen's Kappa is never greater than 1 or equivalent to 1. Values of 0 or less signify an ineffective classifier.

5.1.5 Accuracy

Accuracy is a measure of the ratio of correct predictions. It is computed by dividing the sum of true positives (TP) and true negatives (TN) by the total number of events. The calculation of accuracy can be performed using either Equation (9).

$$\text{Accuracy} = \frac{\text{No. of true predictions}}{\text{Total No. of predictions}}. \tag{9}$$

5.1.6 Matthews Correlation Coefficient (MCC)

It accounts for true and false positives and negatives in machine learning and is widely recognized as a metric that can be applied even when the class sizes vary.

The MCC is a correlation coefficient number between -1 and 1. A flawless forecast has a coefficient of +1, while an average random forecast has a coefficient of 0 and an inverse forecast has a coefficient of -1. Another name for the statistic is the phi coefficient. The MCC can be determined using Equation (10).

$$\text{MCC} = \frac{(\text{TP} * \text{TN}) - (\text{FP} * \text{FN})}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}. \quad (10)$$

5.1.7 Learning Curve

A learning curve is a visual representation of the relationship between a learner's performance and the amount of time necessary to complete an activity. The learning curve for the proposed model is depicted in Figure 7.

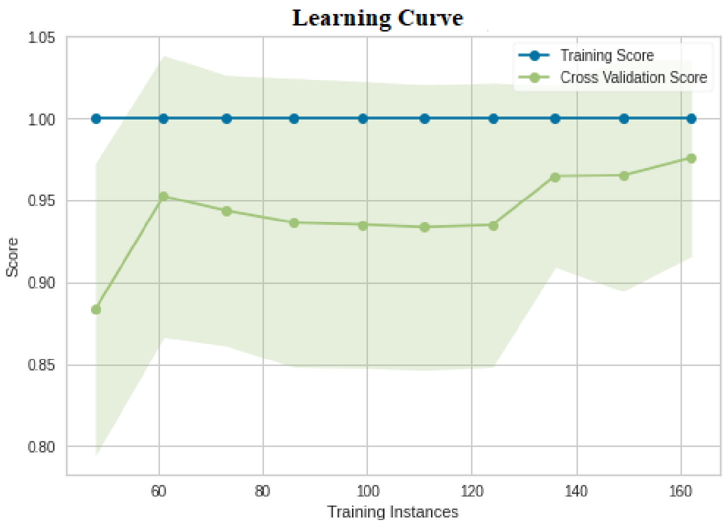

Figure 7. Learning curve of proposed model

5.1.8 K-Fold Cross Validation

Conducting extensive comparisons is a recommended practice for assessing the performance of a model. One way to achieve this is by performing K-fold cross-validation repeatedely, which involves executing K-fold validation multiple times or increasing the number of comparisons, as discussed by Arlot and Celisse in [36]. Inside it, only K comparisons are generated, with random data allocated for each iteration. This technique is employed to gauge the resilience of machine learning models. In our study, we evaluate the robustness of the best prediction model through ten consecutive executions, commonly referred to as 10-fold cross-validation.

5.2 Implementation

5.2.1 Algorithm

Algorithm 1 is used to program the proposed approach. The Load Prediction algorithm provides a mathematical explanation for the phases of data processing and prediction. The conventions used in the algorithm are shown in Table 7. The Load Prediction Algorithm involved examining three case studies, each with a varying number of virtual machines (VMs). For each VM, various features were extracted from the dataset. Next, the dataset was transformed into three categories: overloaded, underloaded, and balanced. Next, the data was divided, and K-fold cross-validation was conducted. There are two segments in the dataset: the training set and the testing set. Seventy percent (70 %) of the data has been allocated for training the models, while the remaining thirty percent (30 %) is reserved for testing. The final step involves employing the ensemble model to obtain the end result.

6 RESULT ANALYSIS, COMPARISON AND DISCUSSION

The performance is evaluated in three different case studies. All the models are evaluated based on three parameters: accuracy, recall and precision.

In Case 1, it has been observed that after applying normalization to the dataset,

Conventions	**Description**
i	Number of cases
j	The number of feature vectors
X, Y	Input and output space
x, y	Input and target vectors
μ	DataSet after preprocessing
$\mu 1$, $\mu 2$	Partitions of the data: Training and testing datasets i.e. $\mu 1$ and $\mu 2$
M	Model Representation i.e., $M : X \to Y$
M_{EM}	Represents the final multilevel ensemble model created
F_{EM}	Final output of ensemble model
e	Acceptable error
V_m, VM	Virtual Machine
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Table 7. Conventions used

The accuracy of models is very similar to the performance of models without preprocessing. However, the accuracy of models decreases after applying PCA to the dataset. NB and KN classifier performance are low compared to all other models, even without preprocessing and with preprocessing of models. However, our

Algorithm 1 Load Prediction Algorithm

```
BEGIN
for $i = 1$ to 3, where i is the number of cases (Count of $V_m$) do
    foreach $V_m = 1$ to n, where n is the number of VMs in case i do
        foreach $V_m$, $\exists x_j \in V_m$, where $x_j$ is collection of features do
            Transformation of data using following equation:
            $\text{Host Status} = \begin{cases} \text{Overloaded}, & \text{if } U_{(i,t)} > T_{ol}, \\ \text{Underloaded}, & \text{if } U_{(i,t)} > T_{ul}, \\ \text{Balanced}, & \text{if } T_{ul} < U_{(i,t)} > T_{ol} \end{cases}$
        Data Partition or split
        K-fold validation ($\mu_1 = \text{random}(\mu, \text{fraction} = 0.70)$)
        $\mu_1 = \mu - \mu_1$
        $\mu_1$ = Training Set
        $\mu 2$ = Testing Set
        ******** Training Models *********
        $M_k$, for $k \in s$. $(s = M_1, M_2 \dots M_n)$
        $M_k[X \to Y] : M_k[x_j \to y_j]$, where $(x_j, y_j) \in \mu_{train}$
        ***** Ensemble and Testing Phase ******
        $F_{EM}(x_j) = M_{EM}(y'_j)$ where $x_j \in \mu_{test}$
        *********** Error Rate **********
        Absolute$(y_j - y'_j) = e$
        Accuracy $= \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$
    Precision $= \frac{\text{TP}}{\text{TP+FP}}$
END
```

proposed ensemble model performance is better than all the applied models. It provides about 78 % accuracy after simultaneously applying outlier removal and normalization to the dataset. Table 8 displays the performance of different models.

In Case 2, it has been observed that the models' accuracy increases after the dataset's normalization and the models' accuracy decreases after applying PCA. NB, Linear Discriminant Analysis, Ridge classifier and K neighbors classifier performance are low compared to all other models when PCA is applied to the dataset. During the normalization of the dataset, four models, the RF model, AB, GB and DT model, perform equally well. However, our proposed ensemble model performance is better than these models, even without preprocessing and with preprocessing of models. Our proposed ensemble model provides maximum accuracy of about 80 % when applying outlier removal and normalization simultaneously to the dataset. Table 9 displays the performance of different models.

In Case 3, it has been observed that models perform more accurately after applying normalization to the dataset. In this case, the RF, GB, AB and DT models perform more accurately than all others. These models also perform better after applying the transformation to the dataset. NB, Ridge classifier, Linear Discriminant classifier and KN classifier performance decrease after applying PCA and feature selection to the dataset. In contrast to these other models, the performance of the suggested ensemble approach is relatively preferable, achieving a maximum accuracy of around 82 % after normalizing the dataset. The performance of different models is shown in Table 10. Figures 9, 8 and 10 illustrate the comparison of various models in terms of their precision, accuracy and recall, respectively.

During the training process, it is possible for models to become biased. The SMOTE algorithm can be employed as a means of addressing this particular issue. Another issue that arises in machine learning is the problems of overfitting and underfitting. To mitigate issues related to over-fitting and under-fitting, it is advisable to utilize cross-validation and assess the model's performance on an independent dataset. Consistent high performance indicates that the model is not affected by over-fitting or under-fitting. Overfitting arises when a model captures an excessive amount of information, whereas underfitting occurs when a model lacks sufficient information.

During cross-validation, the model is repeatedly evaluated n times, and its accuracy is measured accordingly. If the recorded accuracy exhibits significant fluctuations, the model may suffer from issues such as overfitting, underfitting, or bias. In this study, repeated K-fold cross-validation is employed to assess the reliability of the correctness of the suggested model, hence demonstrating its resilience against potential issues [36]. The suggested model exhibits a lack of over-fitting, underfitting, and bias-related concerns. Furthermore, the results of the suggested model demonstrate a significant improvement compared to existing methodologies.

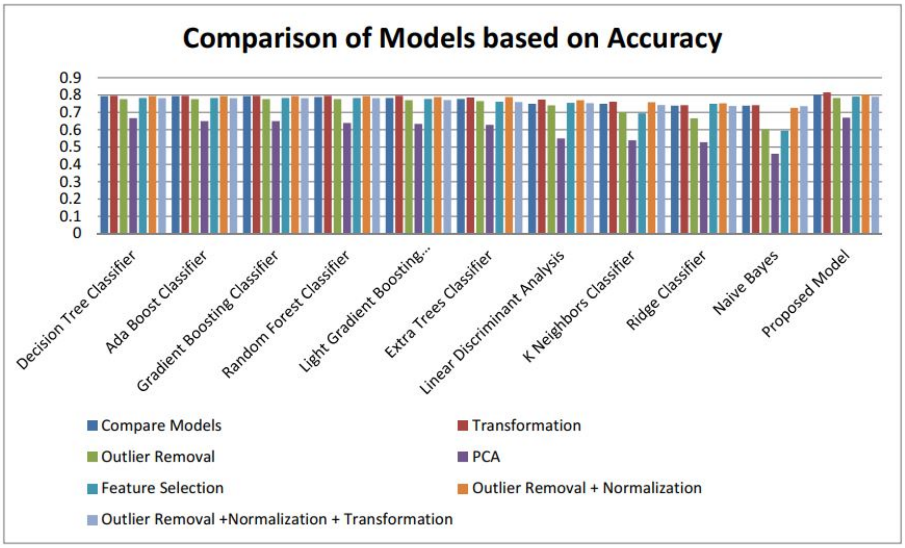

Figure 8. Different models comparison based on accuracy

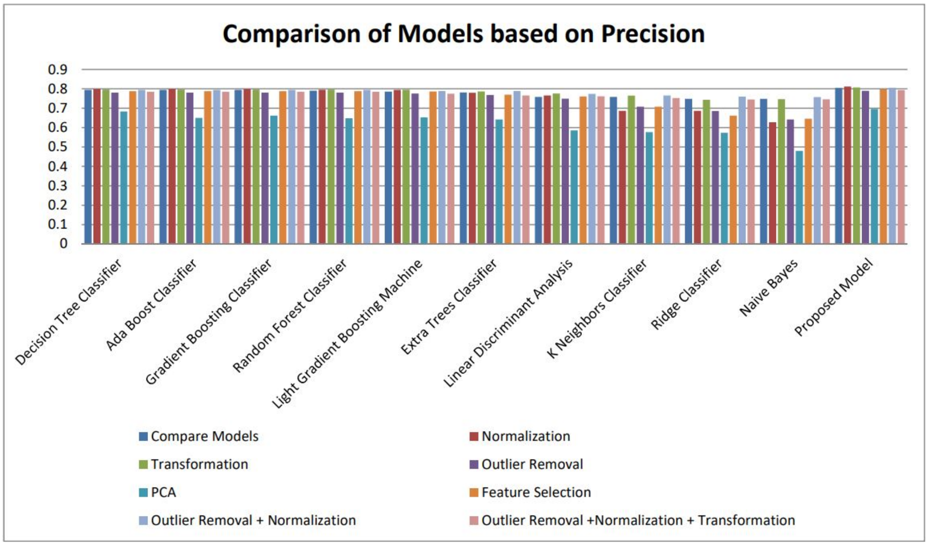

Figure 9. Different models comparison based on precision

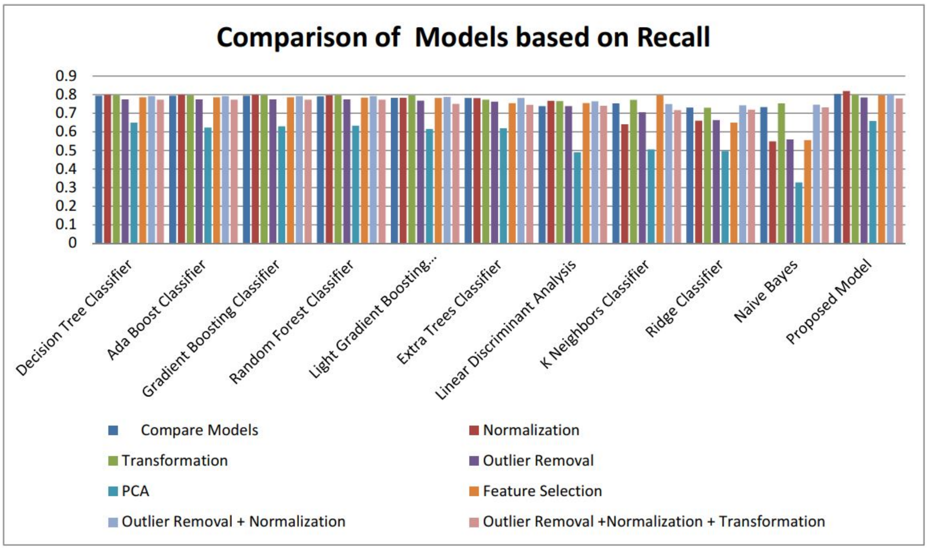

Figure 10. Different models comparison based on recall

		Compare Models			Normalization			Transformation			Outlier Removal		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.760	0.758	0.734	0.757	0.759	0.758	0.754	0.754	0.756	0.726	0.724	0.731
ada	Ada Boost	0.758	0.758	0.734	0.758	0.758	0.758	0.754	0.754	0.755	0.726	0.725	0.731
gbc	Gradient Boosting	0.758	0.759	0.730	0.757	0.757	0.756	0.754	0.754	0.755	0.726	0.725	0.731
rf	Random Forest	0.758	0.757	0.734	0.758	0.757	0.758	0.753	0.754	0.755	0.725	0.728	0.731
lightgbm	Light Gradient Boosting	0.752	0.753	0.752	0.754	0.741	0.755	0.748	0.743	0.749	0.721	0.718	0.625
et	Extra Trees	0.732	0.734	0.733	0.736	0.728	0.737	0.743	0.747	0.746	0.715	0.713	0.718
lda	Linear Discriminant Analysis (LDA)	0.721	0.663	0.714	0.736	0.685	0.718	0.747	0.756	0.721	0.711	0.708	0.709
ridge	Ridge Classifier	0.648	0.534	0.645	0.728	0.726	0.732	0.727	0.764	0.740	0.641	0.676	0.658
knn	K Neighbor Classifier	0.508	0.485	0.515	0.703	0.683	0.708	0.716	0.739	0.740	0.626	0.614	0.645
nb	Naive Bayes	0.355	0.423	0.415	0.728	0.711	0.715	0.716	0.761	0.722	0.551	0.509	0.601
	Proposed Model	0.760	0.771	0.772	0.767	0.768	0.771	0.763	0.772	0.775	0.745	0.735	0.747

		PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
		Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.666	0.649	0.623	0.763	0.745	0.768	0.758	0.774	0.758	0.742	0.723	0.735
ada	Ada Boost	0.65	0.63	0.642	0.763	0.745	0.768	0.758	0.763	0.758	0.742	0.723	0.735
gbc	Gradient Boosting	0.639	0.633	0.649	0.763	0.744	0.764	0.758	0.763	0.756	0.742	0.723	0.735
rf	Random Forest	0.65	0.621	0.650	0.763	0.746	0.768	0.758	0.763	0.758	0.739	0.723	0.735
lightgbm	Light Gradient Boosting	0.633	0.615	0.653	0.757	0.742	0.761	0.763	0.758	0.749	0.720	0.740	0.724
et	Extra Trees	0.628	0.619	0.622	0.741	0.714	0.710	0.763	0.752	0.742	0.705	0.736	0.716
lda	Linear Discriminant Analysis (LDA)	0.515	0.489	0.586	0.736	0.715	0.701	0.710	0.724	0.734	0.704	0.710	0.713
ridge	Ridge Classifier	0.544	0.506	0.577	0.724	0.657	0.708	0.748	0.720	0.719	0.656	0.687	0.693
knn	K Neighbor Classifier	0.529	0.497	0.574	0.815	0.611	0.642	0.713	0.723	0.720	0.677	0.689	0.694
nb	Naive Bayes	0.461	0.327	0.481	0.554	0.546	0.646	0.727	0.717	0.717	0.686	0.672	0.676
	Proposed Model	0.677	0.659	0.631	0.750	0.757	0.793	0.780	0.773	0.782	0.754	0.753	0.749

Table 8. Detailed comparison of models of Case 1

		Compare Models			Normalization			Transformation			Outlier Removal		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.799	0.790	0.791	0.780	0.796	0.793	0.763	0.784	0.795	0.776	0.774	0.781
ada	Ada Boost	0.785	0.789	0.788	0.781	0.790	0.793	0.784	0.771	0.785	0.766	0.785	0.780
gbc	Gradient Boosting	0.786	0.780	0.783	0.791	0.793	0.742	0.754	0.734	0.705	0.776	0.742	0.781
rf	Random Forest	0.796	0.796	0.798	0.795	0.798	0.790	0.794	0.795	0.795	0.776	0.774	0.781
lightgbm	Light Gradient Boosting	0.772	0.734	0.792	0.774	0.782	0.724	0.768	0.773	0.749	0.770	0.768	0.676
et	Extra Trees	0.772	0.774	0.773	0.776	0.769	0.777	0.783	0.787	0.786	0.764	0.763	0.769
lda	Linear Discriminant Analysis (LDA)	0.724	0.703	0.734	0.756	0.745	0.767	0.772	0.775	0.780	0.721	0.739	0.788
ridge	Ridge Classifier	0.680	0.588	0.699	0.748	0.726	0.776	0.768	0.789	0.783	0.701	0.716	0.708
knn	K Neighbor Classifier	0.558	0.515	0.565	0.754	0.733	0.759	0.767	0.769	0.771	0.666	0.664	0.685
nb	Naive Bayes	0.405	0.453	0.465	0.748	0.741	0.750	0.755	0.752	0.761	0.601	0.559	0.642
	Proposed Model	0.80	0.811	0.812	0.818	0.818	0.815	0.803	0.812	0.815	0.785	0.786	0.798

		PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.666	0.649	0.683	0.783	0.786	0.788	0.794	0.793	0.795	0.782	0.773	0.785
ada	Ada Boost	0.652	0.631	0.682	0.780	0.781	0.786	0.734	0.763	0.795	0.732	0.773	0.785
gbc	Gradient Boosting	0.609	0.613	0.642	0.787	0.783	0.781	0.799	0.773	0.745	0.762	0.783	0.780
rf	Random Forest	0.650	0.623	0.652	0.783	0.786	0.788	0.794	0.793	0.795	0.782	0.773	0.785
lightgbm	Light Gradient Boosting	0.633	0.615	0.653	0.778	0.782	0.787	0.788	0.788	0.789	0.771	0.751	0.775
et	Extra Trees	0.628	0.619	0.642	0.761	0.754	0.770	0.788	0.782	0.799	0.759	0.746	0.766
lda	Linear Discriminant Analysis (LDA)	0.550	0.469	0.586	0.745	0.714	0.761	0.750	0.764	0.775	0.753	0.740	0.762
ridge	Ridge Classifier	0.544	0.506	0.577	0.794	0.697	0.708	0.759	0.749	0.766	0.742	0.717	0.753
knn	K Neighbor Classifier	0.528	0.497	0.573	0.850	0.650	0.662	0.753	0.743	0.760	0.737	0.719	0.745
nb	Naive Bayes	0.461	0.327	0.480	0.594	0.556	0.647	0.747	0.747	0.757	0.736	0.732	0.746
	Proposed Model	0.676	0.658	0.691	0.790	0.797	0.803	0.801	0.813	0.827	0.794	0.785	0.796

Table 9. Detailed comparison of models of Case 2

		Compare Models			Normalization			Transformation			Outlier Removal		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.798	0.790	0.789	0.799	0.813	0.815	0.786	0.767	0.786	0.772	0.770	0.783
ada	Ada Boost	0.789	0.788	0.790	0.812	0.810	0.814	0.769	0.790	0.786	0.705	0.715	0.782
gbc	Gradient Boosting (GBC)	0.792	0.763	0.785	0.800	0.799	0.812	0.776	0.787	0.798	0.756	0.775	0.781
rf	Random Forest	0.789	0.785	0.787	0.790	0.791	0.769	0.790	0.779	0.729	0.767	0.747	0.780
lightgbm	Light Gradient Boosting	0.782	0.780	0.798	0.774	0.781	0.775	0.769	0.764	0.798	0.750	0.784	0.762
et	Extra Trees	0.777	0.742	0.782	0.777	0.781	0.780	0.785	0.773	0.786	0.764	0.765	0.769
lda	Linear Discriminant Analysis (LDA)	0.749	0.719	0.728	0.763	0.764	0.768	0.770	0.726	0.767	0.742	0.731	0.745
knn	K Neighbor Classifier	0.750	0.753	0.758	0.672	0.649	0.685	0.769	0.760	0.762	0.701	0.712	0.718
ridge	Ridge Classifier	0.739	0.735	0.747	0.662	0.642	0.686	0.744	0.732	0.741	0.661	0.662	0.684
nb	Naive Bayes	0.718	0.732	0.749	0.590	0.549	0.626	0.742	0.753	0.749	0.600	0.558	0.641
	Proposed Model	0.802	0.804	0.805	0.821	0.824	0.812	0.816	0.801	0.807	0.784	0.784	0.792

		PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.669	0.640	0.681	0.782	0.785	0.780	0.791	0.773	0.784	0.746	0.713	0.745
ada	Ada Boost	0.650	0.623	0.650	0.788	0.784	0.781	0.748	0.797	0.784	0.781	0.713	0.705
gbc	Gradient Boosting (GBC)	0.650	0.631	0.602	0.783	0.786	0.708	0.794	0.798	0.775	0.783	0.773	0.785
rf	Random Forest	0.659	0.683	0.699	0.753	0.744	0.768	0.774	0.783	0.725	0.702	0.763	0.775
lightgbm	Light Gradient Boosting	0.623	0.625	0.643	0.718	0.788	0.771	0.758	0.727	0.769	0.719	0.720	0.785
et	Extra Trees	0.628	0.619	0.641	0.761	0.754	0.770	0.788	0.782	0.789	0.759	0.746	0.766
lda	Linear Discriminant Analysis (LDA)	0.510	0.489	0.586	0.756	0.755	0.761	0.771	0.765	0.774	0.754	0.740	0.762
knn	K Neighbor Classifier	0.534	0.519	0.550	0.654	0.783	0.728	0.752	0.740	0.763	0.724	0.720	0.754
ridge	Ridge Classifier	0.529	0.496	0.573	0.621	0.650	0.662	0.752	0.743	0.720	0.737	0.759	0.745
nb	Naive Bayes	0.460	0.326	0.480	0.595	0.558	0.647	0.748	0.757	0.759	0.738	0.712	0.776
	Proposed Model	0.672	0.658	0.696	0.791	0.796	0.798	0.804	0.803	0.806	0.790	0.781	0.794

Table 10. Detailed comparison of models of Case 3

7 CONCLUSION AND FUTURE WORK

This work introduces an innovative reliability framework that encompasses multiple phases of implementation, beginning with the generation of virtual machines via the command line with multiple random settings, followed by the generation of datasets, machine learning techniques, and result analysis. All models are evaluated for their precision, recall, and accuracy. A total of ten machine learning models were employed in order to construct an ensemble model, which ultimately yielded optimal and accurate results for the classification of host loads. It has been observed that applying normalization to a dataset improves the performance of models. Four models, the RF, AB, GB, and DT models, performed equally well in all three case studies during normalization of the dataset. However, compared to these models, our proposed ensemble model performs marginally better, with an accuracy of approximately 82 %. The robustness of an ensemble model was then evaluated using the 10-fold cross-validation method [36]. Improving energy efficiency in a fog environment is a challenging job. While Container as a Service (CaaS) is gaining increasing popularity, there has been relatively little emphasis on evaluating the energy efficiency of resource management algorithms within this service framework. Future work will include migrating containers from one virtual machine to another, which will only occur when a host is determined to be overloaded or underloaded based on predetermined criteria. We plan to improve the energy consumption of hosts in fog environment by migrating containers in the future.

REFERENCES

[1] Atzori, L.—Iera, A.—Morabito, G.: The Internet of Things: A Survey. Computer Networks, Vol. 54, 2010, No. 15, pp. 2787–2805, doi: 10.1016/j.comnet.2010.05.010.

[2] Ning, H.—Liu, H.—Ma, J.—Yang, L. T.—Huang, R.: Cybermatics: Cyber–Physical–Social–Thinking Hyperspace Based Science and Technology. Future Generation Computer Systems, Vol. 56, 2016, pp. 504–522, doi: 10.1016/j.future.2015.07.012.

[3] Cortés, R.—Bonnaire, X.—Marin, O.—Sens, P.: Stream Processing of Healthcare Sensor Data: Studying User Traces to Identify Challenges from a Big Data Perspective. Procedia Computer Science, Vol. 52, 2015, pp. 1004–1009, doi: 10.1016/j.procs.2015.05.093.

[4] Hu, P.—Dhelim, S.—Ning, H.—Qiu, T.: Survey on Fog Computing: Architecture, Key Technologies, Applications and Open Issues. Journal of Network and Computer Applications, Vol. 98, 2017, pp. 27–42, doi: 10.1016/j.jnca.2017.09.002.

[5] Yi, S.—Hao, Z.—Qin, Z.—Li, Q.: Fog Computing: Platform and Applications. 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2015, pp. 73–78, doi: 10.1109/HotWeb.2015.22.

[6] Pereira, E.—Fischer, I. A.—Medina, R. D.—Carreno, E. D.—Padoin, E. L.: A Load Balancing Algorithm for Fog Computing Environments.

In: Crespo-Mariño, J. L., Meneses-Rojas, E. (Eds.): High Performance Computing (CARLA 2019). Springer, Cham, Communications in Computer and Information Science, Vol. 1087, 2019, pp. 65–77, doi: 10.1007/978-3-030-41005-6_5.

[7] Ahmad, R. W.—Gani, A.—Hamid, S. H. A.—Shiraz, M.—Yousafzai, A.—Xia, F.: A Survey on Virtual Machine Migration and Server Consolidation Frameworks for Cloud Data Centers. Journal of Network and Computer Applications, Vol. 52, 2015, pp. 11–25, doi: 10.1016/j.jnca.2015.02.002.

[8] Fan, X.—Weber, W. D.—Barroso, L. A.: Power Provisioning for a Warehouse-Sized Computer. ACM SIGARCH Computer Architecture News, Vol. 35, 2007, No. 2, pp. 13–23, doi: 10.1145/1273440.1250665.

[9] Barroso, L. A.—Hölzle, U.: The Case for Energy-Proportional Computing. Computer, Vol. 40, 2007, No. 12, pp. 33–37, doi: 10.1109/MC.2007.443.

[10] Abdelsamea, A.—El-Moursy, A. A.—Hemayed, E. E.—Eldeeb, H.: Virtual Machine Consolidation Enhancement Using Hybrid Regression Algorithms. Egyptian Informatics Journal, Vol. 18, 2017, No. 3, pp. 161–170, doi: 10.1016/j.eij.2016.12.002.

[11] Liu, X.—Wu, J.—Sha, G.—Liu, S.: Virtual Machine Consolidation with Minimization of Migration Thrashing for Cloud Data Centers. Mathematical Problems in Engineering, Vol. 2020, 2020, Art. No. 7848232, doi: 10.1155/2020/7848232.

[12] Piraghaj, S. F.—Dastjerdi, A. V.—Calheiros, R. N.—Buyya, R.: ContainerCloudSim: An Environment for Modeling and Simulation of Containers in Cloud Data Centers. Software: Practice and Experience, Vol. 47, 2017, No. 4, pp. 505–521, doi: 10.1002/spe.2422.

[13] Huang, H. C.—Cressie, N.: Spatio-Temporal Prediction of Snow Water Equivalent Using the Kalman Filter. Computational Statistics & Data Analysis, Vol. 22, 1996, No. 2, pp. 159–175, doi: 10.1016/0167-9473(95)00047-X.

[14] Calheiros, R. N.—Masoumi, E.—Ranjan, R.—Buyya, R.: Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. IEEE Transactions on Cloud Computing, Vol. 3, 2015, No. 4, pp. 449–458, doi: 10.1109/TCC.2014.2350475.

[15] Chehelgerdi-Samani, M.—Safi-Esfahani, F.: PCVM.ARIMA: Predictive Consolidation of Virtual Machines Applying ARIMA Method. The Journal of Supercomputing, Vol. 77, 2021, No. 3, pp. 2172–2206, doi: 10.1007/s11227-020-03354-3.

[16] Dhaval, B.—Deshpande, A.: Short-Term Load Forecasting Using Method of Multiple Linear Regression. In: Balo, F. (Ed.): New Approaches in Engineering Research Vol. 14. BP International, 2021, pp. 67–77, doi: 10.9734/bpi/naer/v14/13047D.

[17] Kaur, M.—Kaur, P. D.—Sood, S. K.: Energy Efficient IoT-Based Cloud Framework for Early Flood Prediction. Natural Hazards, Vol. 109, 2021, No. 3, pp. 2053–2076, doi: 10.1007/s11069-021-04910-7.

[18] Tan, M. C.—Wong, S. C.—Xu, J. M.—Guan, Z. R.—Zhang, P.: An Aggregation Approach to Short-Term Traffic Flow Prediction. IEEE Transactions on Intelligent Transportation Systems, Vol. 10, 2009, No. 1, pp. 60–69, doi: 10.1109/TITS.2008.2011693.

[19] Janardhanan, D.—Barrett, E.: CPU Workload Forecasting of Machines in Data Centers Using LSTM Recurrent Neural Networks and ARIMA Models. 2017 12th In-

ternational Conference for Internet Technology and Secured Transactions (ICITST), IEEE, 2017, pp. 55–60, doi: 10.23919/ICITST.2017.8356346.

[20] Fan, J.—Zhang, K.—Huang, Y.—Zhu, Y.—Chen, B.: Parallel Spatio-Temporal Attention-Based TCNN for Multivariate Time Series Prediction. Neural Computing and Applications, Vol. 35, 2023, No. 18, pp. 13109–13118, doi: 10.1007/s00521-021-05958-z.

[21] Kumar, J.—Singh, A. K.—Buyya, R.: Self Directed Learning Based Workload Forecasting Model for Cloud Resource Management. Information Sciences, Vol. 543, 2021, pp. 345–366, doi: 10.1016/j.ins.2020.07.012.

[22] Khan, T.—Tian, W.—Ilager, S.—Buyya, R.: Workload Forecasting and Energy State Estimation in Cloud Data Centers: ML-Centric Approach. Future Generation Computer Systems, Vol. 128, 2022, pp. 320–332, doi: 10.1016/j.future.2021.10.019.

[23] Patel, Y. S.—Misra, R.: Performance Comparison of Deep VM Workload Prediction Approaches for Cloud. In: Pattnaik, P. K., Rautaray, S. S., Das, H., Nayak, J. (Eds.): Progress in Computing, Analytics and Networking. Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 710, 2018, pp. 149–160, doi: 10.1007/978-981-10-7871-2_15.

[24] Gao, J.—Wang, H.—Shen, H.: Machine Learning Based Workload Prediction in Cloud Computing. 2020 29th International Conference on Computer Communications and Networks (ICCCN), IEEE, 2020, pp. 1–9, doi: 10.1109/ICCCN49398.2020.9209730.

[25] Kumar, J.—Singh, A. K.: Workload Prediction in the Cloud Using Artificial Neural Network and Adaptive Differential Evolution. Future Generation Computer Systems, Vol. 81, 2018, pp. 41–52, doi: 10.1016/j.future.2017.10.047.

[26] Ramezani, F.—Naderpour, M.: A Fuzzy Virtual Machine Workload Prediction Method for Cloud Environments. 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017, pp. 1–6, doi: 10.1109/FUZZ-IEEE.2017.8015450.

[27] Yang, J.—Liu, C.—Shang, Y.—Cheng, B.—Mao, Z.—Liu, C.—Niu, L.—Chen, J.: A Cost-Aware Auto-Scaling Approach Using the Workload Prediction in Service Clouds. Information Systems Frontiers, Vol. 16, 2014, No. 1, pp. 7–18, doi: 10.1007/s10796-013-9459-0.

[28] Qiu, F.—Zhang, B.—Guo, J.: A Deep Learning Approach for VM Workload Prediction in the Cloud. 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016, pp. 319–324, doi: 10.1109/SNPD.2016.7515919.

[29] Jheng, J. J.—Tseng, F. H.—Chao, H. C.—Chou, L. D.: A Novel VM Workload Prediction Using Grey Forecasting Model in Cloud Data Center. The International Conference on Information Networking 2014 (ICOIN2014), IEEE, 2014, pp. 40–45, doi: 10.1109/ICOIN.2014.6799662.

[30] Yu, Y.—Jindal, V.—Yen, I. L.—Bastani, F.: Integrating Clustering and Learning for Improved Workload Prediction in the Cloud. 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 876–879, doi: 10.1109/CLOUD.2016.0127.

[31] Eramo, V.—Catena, T.: Application of an Innovative Convolutional/LSTM Neural Network for Computing Resource Allocation in NFV Network Architectures. IEEE Transactions on Network and Service Management, Vol. 19, 2022, No. 3, pp. 2929–2943, doi: 10.1109/TNSM.2022.3142182.

[32] Caron, E.—Desprez, F.—Muresan, A.: Forecasting for Grid and Cloud Computing on-Demand Resources Based on Pattern Matching. 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 2010, pp. 456–463, doi: 10.1109/CloudCom.2010.65.

[33] Roy, N.—Dubey, A.—Gokhale, A.: Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 500–507, doi: 10.1109/CLOUD.2011.42.

[34] Yadav, R.—Zhang, W.—Li, K.—Liu, C.—Laghari, A. A.: Managing Overloaded Hosts for Energy-Efficiency in Cloud Data Centers. Cluster Computing, Vol. 24, 2021, No. 3, pp. 2001–2015, doi: 10.1007/s10586-020-03182-3.

[35] Whig, P.—Gupta, K.—Jiwani, N.—Jupalle, H.—Kouser, S.—Alam, N.: A Novel Method for Diabetes Classification and Prediction with Pycaret. Microsystem Technologies, Vol. 29, 2023, No. 10, pp. 1479–1487, doi: 10.1007/s00542-023-05473-2.

[36] Arlot, S.—Celisse, A.: A Survey of Cross-Validation Procedures for Model Selection. Statistics Surveys, Vol. 4, 2010, pp. 40–79, doi: 10.1214/09-SS054.

Shabnam Bawa received her B.Tech. degree in computer science and engineering from the Guru Kashi University, Talwandi Sabo and her M.Tech. degree in computer science and engineering from the Punjabi University, Patiala, India. Presently, she is pursuing her Ph.D. in the Computer Science and Engineering Department at the Thapar Institute of Engineering and Technology, Patiala, India. Her research interests include machine learning, computational problems, natural language processing, deep learning and networking.

Prashant Singh Rana is Associate Professor at the Department of Computer Science, Thapar Institute of Engineering and Technology, Patiala, India. He also worked as Project Scientist at IIT Delhi, India. He received his Ph.D. from ABV-Indian Institute of Information Technology and Management, Gwalior, India and his areas of research are machine learning, soft computing, combinatorial problems and computational biology.

Rajkumar Tekchandani is Assistant Professor in the Department of Computer Science and Engineering at Thapar Institute of Engineering and Technology, Patiala. He has nine years of academic experience, including one year of academic experience at NIT Jalandhar. He received his M.Tech. degree in computer science from the Jaypee University of Information Technology, Waknaghat. He received his Ph.D. from the Thapar Institute of Engineering and Technology and his areas of research are software engineering, software code clone detection and data mining.