Cluster Computing (2024) 27:5699–5719
https://doi.org/10.1007/s10586-023-04258-6

An effective deep learning architecture leveraging BIRCH clustering for resource usage prediction of heterogeneous machines in cloud data center

Sheetal Garg[1] · Rohit Ahuja[1] · Raman Singh[2] · Ivan Perl[3]

Received: 11 September 2023 / Revised: 7 December 2023 / Accepted: 21 December 2023 / Published online: 6 February 2024
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

Abstract

Given the rise in demand for cloud computing in the modern era, the effectiveness of resource utilization is eminent to decrease energy footprint and achieve economic services. With the emerging machine learning and artificial intelligence techniques to model and predict, it is essential to explore a principal method that provides the best solution for the accurate provisioning of forthcoming requests in a cloud data center. Recent studies used machine learning and other advanced analytics to predict resource usage; however, these do not consider long-range dependencies in the time series, which is essential to capture for better prediction. Further, they show limitations in handling noise, missing values, and outliers in datasets. In this paper, we explored the problem by studying three techniques that enabled us to answer improvements in short-term forecasting of physical machines' resource usage if the above factors are considered. We evaluated the predictions using Transformer and Informer deep learning models that cover the above aspects and compared them with the Long short-term memory (LSTM) model. We used a real-world Google cluster trace usage dataset and employed Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm to select heterogeneous machines. The evaluation of the three models depicts that the Transformer architecture that considers long-range dependencies in time series and shortcomings with datasets shows improvement in forecasting with 14.2% reduction in RMSE than LSTM. However, LSTM shows better results for some machines than the Transformer, which depicts the importance of input sequence order. The Informer model, which considers both dependencies and is a hybrid of LSTM and Transformer, outperformed both models with 21.7% from LSTM and 20.8% from Transformer reduction in RMSE. The results also depict Informer model consistently performs better than the other models across all subsets of the dataset. Our study proves that considering long-range dependencies and sequence ordering for resource usage time series improves the prediction.

Keywords Time-series prediction · Clustering · LSTM · Transformer · Informer · Cloud computing

✉ Sheetal Garg
sgarg_phd19@thapar.edu

Rohit Ahuja
rohit.ahuja@thapar.edu

Raman Singh
raman.singh@uws.ac.uk

Ivan Perl
ivan.perl@itmo.ru

[1] Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, Punjab 147004, India

[2] School of Computing, Engineering and Physical Sciences, University of the West of Scotland, United Kingdom, Hamilton, Scotland G72 0LH, UK

[3] Faculty of Software Engineering and Computer Systems, ITMO University, Saint Petersburg 197101, Russia

1 Introduction

Cloud computing has become integral to modern IT infrastructure due to its scalability and flexibility, enabling organizations to quickly and conveniently acquire computing resources based on their dynamic business needs. With growing demand, a cloud service provider must strive for infrastructure capacity planning and effective resource allocation for improved service performance and availability. Therefore, accurate resource usage prediction is crucial information for resource allocation, which helps in better managing the organization, improving system performance, and reducing operational costs [1–4].

Recent studies on resource usage prediction used statistical and machine-learning approaches. ARIMA [5–7], SARIMA [8–10], Exponential Smoothing [11–13], Moving average [14], Random Forest [15, 16], Support Vector Machines [17, 18], Decision Trees [18], Neural Networks [19–21], Autoencoders [22, 23], and Long Short-Term Memory (LSTM) are widely employed for time series prediction [8, 24, 25]. However, these have a limited ability to capture non-linear trends and long-range dependencies. Further, they are sensitive to outliers, difficult to parallelize for fast processing, and tend to overfit noisy data [26–28]. Thus, It is vital for a model to handle the above shortcomings and should be tested on an extensive heterogenous representative sample of physical machines in a large dataset for robust evaluation.

Transformer, a deep learning-based architecture, has recently gained importance due to its ability to capture the different parts of an input sequence. This model is based on self-attention mechanisms that allow it for efficient parallel processing and training on the long-range dependencies of a time series [29]. Such a behaviour was missing in RNN and convolutional neural networks (CNNs) like LSTM. These are trained sequentially and on backpropagation through time (BPTT), which can suffer from vanishing and exploding gradients [30]. However, in time series prediction, especially in resource usage of a physical machine in a cloud environment, the input sequence order over time steps is an important consideration. During training, LSTM takes input at the current time step and the hidden state from the previous time step. The output at each time step is the sequence predicted value for the next observation. Thus, LSTM offers an advantage by preserving the input order over Transformer, which is much more effective when the order is unimportant. Informer, another deep learning algorithm, offers both benefits by combining a self-attention mechanism as in Transformer and a convolutional layer as in LSTM. This allows the model to capture local and global (long-range) dependencies and makes it suitable for time series prediction [31].

Accurately predicting the resource usage of physical machines in the cloud is essential for efficient cloud management. This practice facilitates capacity planning, cost optimization, and performance enhancement. By accurately forecasting resource needs, organizations can avoid overprovisioning, optimize energy consumption, and meet service-level agreements [32]. Predictive analytics also aids in fault detection, prevention, and contributes to a better user experience. Additionally, it enables auto-scaling mechanisms, ensuring that cloud resources dynamically adjust to meet fluctuating workloads, leading to improved overall reliability and responsiveness [33].

In this paper, we designed an experiment in which we evaluated three models, LSTM, Transformer, and Informer, in a cloud computing environment to evaluate and compare them with each other by predicting the CPU usage of physical machines. We utilized a large real-world Google cluster trace usage dataset comprising information regarding job scheduling, resource utilization, and power usage of Google's data centre [34]. An opportunistic but challenging aspect of a large dataset is to extract representative and heterogeneous information for the best analysis and performance evaluation of the models. Clustering allows us to reduce the data's complexity by grouping similarities and extracting representative samples. In this research, we employed Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm [35] to rationally select a set of physical machines from each cluster based on CPU, memory, and hard disk usage from the entire dataset. This assists in the robust analysis of models' performance on heterogeneous complex resource usage patterns. The contributions of the suggested approach can be summed up as follows:

- We presented an analysis to identify a best-suited model for time-series resource usage prediction in a cloud environment by understanding the influence of long-range dependencies and input sequence order on accuracy. Our contribution systematically compares the existing and widely used LSTM with Transformer and Informer models. The study shows the application of later models for the first time in resource usage prediction.
- We identified the optimal hyperparameters for each model by a randomized search.
- We presented the methodology of extracting representative samples of heterogeneous physical machines' data considering CPU, memory and hard disk usage attributes from the Google cluster usage trace dataset for robust evaluation of models.
- We identified the best window size with minimum root mean square error by performing experiments with different window sizes.

The rest of the work is organized in the following manner. Section 2 describes related studies on time-series based workload prediction. Section 3 describes the theoretical background of applied models. Section 4 describes the suggested methodology, which includes data pre-processing, selection of heterogeneous machines, and model implementation. Finally, Sects. 5 and 6 present the results and discussion and conclusion.

2 Related work

Traditional time series forecasting approaches merely consider linear relationships among variables and neglect the complex nonlinear dynamics. Due to the above limitations, their outcomes are substandard and ineffective for learning intricate patterns. Time series-based workload prediction is categorized by leveraging three models, i.e., Neural network-based time series prediction, Transformer model-based time series prediction, and Informer model-based time series prediction. Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM), Transformer, and Informer models can find the nonlinear dependencies between variables to learn a complex pattern and give better performance as compared to traditional models.

2.1 Neural network-based time series prediction

Minxian Xu et al. [36] introduced an efficient supervised learning deep neural network (esDNN) to predict cloud workload. Firstly, multivariate data is converted into supervised learning for deep learning processing using the sliding window technique. Afterwards, Gated Recurrent Unit was employed to achieve accurate prediction. esDNN was validated using Alibaba as well as Google cluster data center. The results of esDNN approach are better than state-of-the-art baseline models. This approach helps to reduce the number of active hosts in a cloud data center. Amine Mrhari et al. [37] designed a model to predict dynamic cloud workload, which is a combination of Variational Mode Decomposition (VMD) and Temporal Convolutional Network (TCN). VMD decomposes the historical data into sequences and provides generated sequences as input to TCN. Habte Lejebo Leka et al. [38] employed a CNN and LSTM neural network for accurate workload prediction. Nhat- Minh et al. [39] offer a time series prediction framework using bidirectional long short-term memory (BiLSTM). The introduced framework focused on multivariate data features and concluded that the multivariate root mean square error was 1.84 times smaller than the univariate BiLSTM. MD. Ebtidaul Karim et al. [40] discussed a model named BHyPrec, a hybrid of the Recurrent neural network models. The hybrid model aggregates LSTM with BiLSTM and a Gated recurrent unit to predict cloud virtual machine CPU usage. This model enhances the prediction capability compared to their separate model's capabilities. Most of the research focused on either seasonal or trend patterns. Anupama K C et al. [8] focused on both kinds of patterns. They employed Seasonal Auto-Regressive Integrated Moving Average, Auto-Regressive Integrated Moving Average, and LSTM for seasonal and non-seasonal workload prediction. Yonghua Zhu et al. [41] introduced an approach to improve workload prediction accuracy. The method employed an LSTM encoder-decoder along with an attention mechanism. The encoder is employed to extract sequential and contextual features from the dataset. Afterwards, a decoder conjunct with an attention mechanism is used to predict workload. Soukaina Ouhame et al. [25] employed CNN and LSTM for multivariate workload resources such as memory, central processing unit, and network usage. Firstly linear dependency is filtered using vector autoregression. Then, CNN takes residual data as input to extract complex features, and the LSTM model has been employed to predict workload. This model enhanced the accuracy rate by 3.8% to 10.9%. It also reduced the error percentage rate by 7% to 8.5% compared to ARIMA-LSTM, AVR-GRU, and VAR-MLP models. Hoang Minh Nguyen et al. [42] offered a long-term short-term memory encoder-decoder (LSTM-ED) method to predict multi-step mean and actual load ahead. The proposed method results concluded that LSTM-ED achieves higher accuracy than LSTM. Eva Patel et al. [43] introduced a hybrid pCNN-LSTM prediction approach. This approach comprises 1D CNN layers with LSTM. CNN was employed to extract patterns from noisy data, and the LSTM layer learned the temporal relationship between raw usage values and extracted data from CNN. The introduced model achieved 15%, 16%, and 13% improvement with Google cluster trace, Bit brain, and Ali baba trace, respectively, to validate the approach over LSTM, BiLSTM, CNN-LSTM and CNN-BLSTM models.

Neural network-based models suffer from "short-term memory" over long sequences like LSTM or LSTM-based encoder-decoder models. However, they do not work appropriately for lengthy sequences [44]. The last unit of LSTM may not capture the complete essence of the sequence. If models are subsequently combined with additional techniques, training time will increase, which makes it challenging to anticipate resource usage in a reasonable timeframe [45].

2.2 Transformer-based time series prediction

The Transformer model is a time series forecasting method based on an encoder-decoder structure. It leverages the

attention mechanism to speed up model training time, making it more suitable for parallel processing and more accurate and efficient than RNN. The unique output mechanism of the transformer model during forecasting reduces error significantly [46].

Ashish Vaswani et al. [47] introduced a network architecture named Transformer. The Transformer model gets aside with recurrence and convolutions entirely and uses attention mechanisms to develop a global relationship between input and output. This model gives one step output at a time. The general architecture of the Transformer employs stacked self-attention and point-wise, fully connected layers for the encoder and decoder. Neo Wu et al. [29] offered a time series forecasting approach that used a transformer-based machine learning model. This methodology employs self-attention mechanisms to extract complex patterns from data collected over time. It also provides a general framework for time series data that incorporates univariate and multivariate parameters. To validate the influenza-like illness dataset, it concluded that their results were better than ARIMA, LSTM, and Seq2Seq+attn models. Binh Tang et al. [48] introduced a deep probabilistic approach that combined the State-space models (SSMs) system with Transformer architecture. In contrast to SSMs, this technique ignores recurrent neural networks and simulates non-Markovian dynamics in the higher dimensional space via an attention mechanism. An additional layer is attached to hierarchically organized stochastic variables to improve the model's expressiveness. Introduced models consistently outperform competitive baselines on several tasks and datasets, including human motion prediction and time series forecasting. These models performed well, but the majority of point prediction models can barely capture the randomness of data.

To resolve these issues, Sifan Wu et al. [49] discussed Adversarial Sparse Transformer (AST) time series forecasting model. This model is based on Generative Adversarial Networks. R. Mohammadi Farsani et al. [50] introduced a prediction method that helps to predict more accurately for larger intervals than existing approaches. They employed encoder-decoder architecture with a self-attention mechanism, i.e., the Transformer model. The introduced model reduces computation complexity and long-term estimation compared with other well-known methods with estimated accuracy, which is up to eight times more resistant and provides a 20% improvement. Recent research has proven the Transformer's ability to enhance prediction. The Transformer isn't directly relevant to long-sequence time-series forecasting due to several significant shortcomings, including quadratic temporal complexity, high memory consumption, and design restrictions of the encoder-decoder architecture.

2.3 Informer-based time series prediction

To overcome Transformer shortcomings, Haoyi Zhou et al. [31] introduced an efficient Informer model for long short time-series forecasting consisting of a Multi-Attention Mechanism, ProbSparse Self-Attention and Encoder-Decoder Architecture. The experimental results concluded that the Informer model significantly improves long sequence data's time series prediction capability. Yufan Qian et al. [51] discussed an adversarial network model called the Informer-WGAN based on the self-attention mechanism. This model can effectively solve the multidimensional time series imputation issue with the support of a discriminator network and training strategy of arbitrary missing rates. The experiment results showed the Informer-WGAN model produces better approximation results than the basic Informer model. Studies show that the problem of error accumulation occurs in time series forecasting approaches based on sliding window forecasting for motor bearing vibration. To resolve the error accumulation problem, Zhengqiang Yang et al. [45] employed the Informer model to forecast motor bearing vibration time series. Informer performs exceptionally well in time series forecasting of motor bearing vibration compared to other forecasting models using publicly available datasets. Li Guo et al. [52] proposed an improved stacked Informer network based on data-driven. The method is mainly employed to predict sequences of electrical line trip faults. To successfully extract underlying features of long sequence time-series data, this method constructs a stacked Informer network and substitutes the original Informer network using adam optimizer with gradient centralized (GC) technology. The results concluded that the proposed method improved sequence prediction accuracy.

3 Theoretical background

This section describes three deep learning models applied for resource usage prediction of physical machines in a cloud data center.

3.1 LSTM

LSTM is a type of recurrent neural network (RNN) architecture commonly used for time series prediction (TSP). It is essential to remember the previous words to predict the next term; therefore, RNN takes input sequences using a loop structure, as represented in Fig. 1a. One limitation of RNNs is that they can struggle to capture long-term dependencies in time series data, as the gradient can become vanishingly small or explode over time [53].

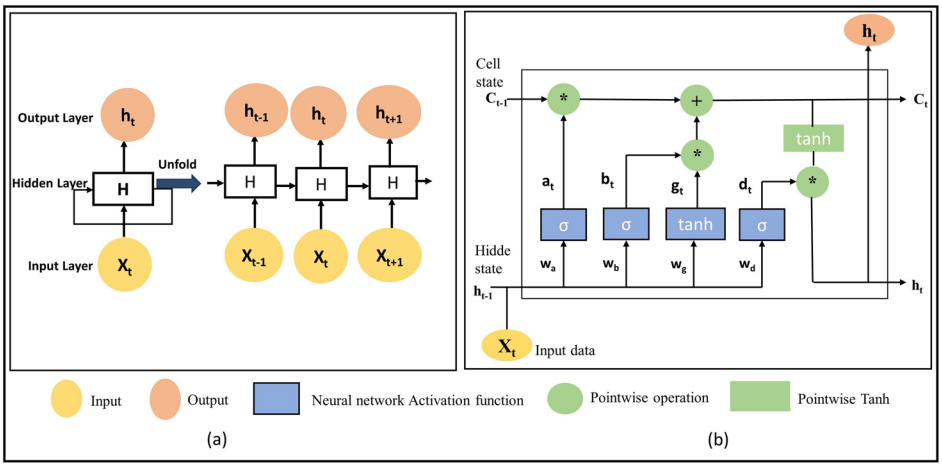


Fig. 1 **a** Architecture of Recurrent neural network. **b** Architecture of Long short term memory model

To address this issue, LSTM has been developed. LSTM is designed to model sequential data by selectively remembering or forgetting information from previous inputs. The LSTM network uses a series of memory cells, gates, and activation functions to learn and predict patterns in sequential data [54]. The mathematical representation of an LSTM cell can be expressed as follows:

First, the input to the LSTM cell is a vector $x(t)$ at time t. The input is transformed by applying a linear transformation using a weight matrix W and a bias vector b:

$$a_t = \sigma(W_a\,[x_t, h_{t-1}] + b_a) \tag{1}$$

$$b_t = \sigma(W_b\,[x_t, h_{t-1}] + b_b) \tag{2}$$

$$d_t = \sigma(W_d\,[x_t, h_{t-1}] + b_d) \tag{3}$$

Here, a_t,b_t, and d_t are the forget, input, and output gates, respectively. They control the flow of information in and out of the memory cell. They are computed by applying a sigmoid activation function to a linear combination of the input vector x_t, the previously hidden state h_{t-1}, and bias terms b_a, b_b, and b_d as depicted in Fig. 1b.

$$g_t = \tanh(W_g\,[x_t, h_{t-1}] + b_g) \tag{4}$$

g_t is candidate value, which is computed by applying the tanh activation function of neural network to a linear combination of input vector x_t, previously hidden state h_{t-1}, and bias term b_g. This new candidate is used to update the memory cell state. Next, the memory cell state C_t is updated by selectively forgetting or remembering information from the previous memory cell state C_{t-1}, current input x_t, based on values of forget gate a_t and the input gate b_t:

$$C_t = a_t * C_{t-1} + b_t * g_t \tag{5}$$

Finally, the hidden state h_t is computed by applying the output gate d_t to the memory cell state C_t, and passing the result through the tanh operation:

$$h_t = d_t * \tanh(C_t) \tag{6}$$

The output of the LSTM cell is the hidden state h_t at time t. This hidden state can be used for prediction or passed to the next LSTM cell in the sequence.

3.2 Transformer

The transformer is based on a self-attention mechanism that allows the model to weigh the importance of different parts of the input sequence when making predictions [47]. This model consists of two main components: the encoder and the decoder. The encoder processes the input sequence and generates a hidden representation, which is then passed to the decoder to generate the output sequence as shown in Fig. 2.

The encoder layer is built up of several identical layers that are layered on top of one another. Each layer has two sublayers: a self-attention layer and a feed-forward layer. The self-attention layer is responsible for weighing the importance of different parts of the input sequence, while the feed-forward layer applies a non-linear transformation to the output of the self-attention layer. The self-attention mechanism is defined by the equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{7}$$

The Q (Query matrix) comprises queries derived from the input data. These queries serve the purpose of retrieving information from key-value pairs, where each row of Q represents a vector corresponding to the current input. The K (Key matrix) represents keys, also derived from the input data in a manner similar to queries. Key vectors play a vital role in determining the degree of attention allocated to different segments of the input sequence when making predictions for a specific time step. Each row of K represents a key vector. The V (Value matrix) represents values,

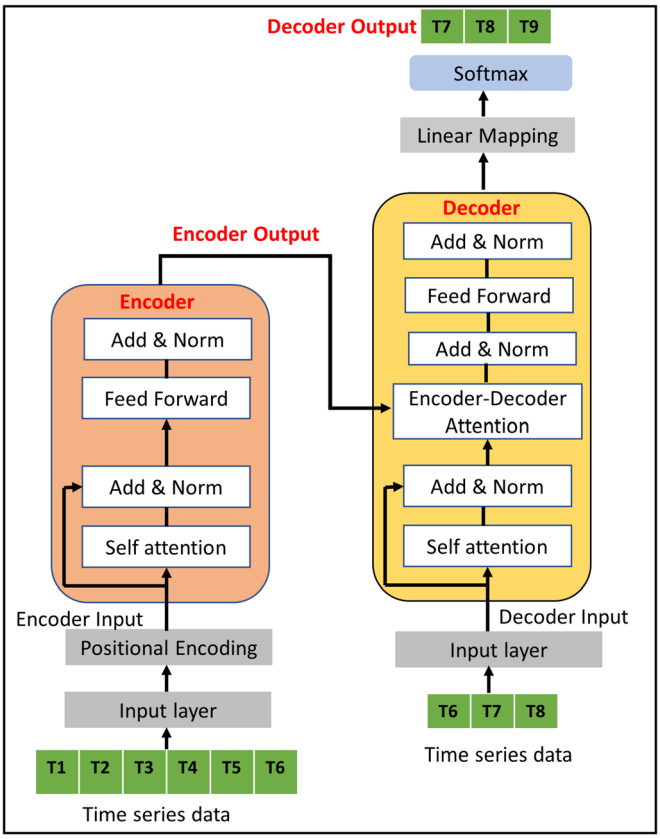


Fig. 2 Architecture of transformer model

also derived from the input data. Key vectors encapsulate information from each position in the input sequence, which the model learns and utilizes for predictions. Each row of V corresponds to a value vector.The Dimensionality of the key vector, denoted as d_k, signifies the dimensionality of the key vectors. It is a scalar value representing this dimensionality. In the Eq. 7, d_k is employed to scale the dot product of Q and K before the application of the softmax function. Dividing by the square root of d_k serves the dual purpose of stabilizing the training process and controlling the magnitude of the dot products.

The resulting weights are then used to compute a weighted sum of the values. After the self-attention layer, the output is passed through a feed-forward layer. The feed-forward layer is defined as follows:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{8}$$

Here, x is the input to the feed-forward layer, W_1, b_1, W_2, and b_2 are learned parameters, and $\max(0, x)$ is the ReLU activation function.

Similarly, each layer in the decoder also consists of two sub-layers: a multi-head self-attention mechanism and a multi-head attention mechanism that attends over the encoder's hidden representation. The output of these sub-layers is passed through a feed-forward network. In addition, it includes a positional encoding mechanism that injects information about the position of each token in the input sequence into the model using Eq. 9 and 10.

$$PE(pos, 2i) = sin\left(\frac{pos}{2L^{\left(\frac{2i}{d}\right)}}\right) \tag{9}$$

$$PE(pos, 2i+1) = sin\left(\frac{pos}{2L^{\left(\frac{2i}{d}\right)}}\right) \tag{10}$$

where pos is the token's position in the input sequence, i is the dimension index, and d_{model} is the dimensionality of the hidden representations. Lastly, to ensure that the prediction of a time series data point will solely depend on earlier data points, the decoder uses look-ahead masking and a one-position offset between the decoder input and target output.

3.3 Informer

The Informer model was introduced to overcome the shortcomings of the Transformer model with three distinct characteristics [31]. (i) The ProbSparse self-attention mechanism is used to replace canonical self-attention, achieving a time complexity and memory usage of $O(L \log L)$, where L represents the input sequence length. This complexity is a significant improvement compared to the traditional $O(L^2)$ complexity that is associated with canonical self-attention. The reduction in complexity is

accomplished through the introduction of probabilistic sparsity, wherein only a subset of elements is considered during attention computation. This innovative design choice enables the model to efficiently handle long input sequences, making the Informer model well-suited for scenarios where computational efficiency is paramount. In addition, the ProbSparse self-attention mechanism enhances the scalability of the architecture, enabling the model to efficiently handle sequences of different lengths while achieving better performance. (ii) By reducing cascade layer input in half, the self-attention distillation effectively handles extremely long input sequences while highlighting dominating attention. (iii) the generative style decoder predicts long time-series sequences in a single forward operation rather than a step-by-step manner, significantly enhancing inference speed for long-sequence predictions. The Informer architecture comprises two major parts: encoder and decoder, as shown in Fig. 3.

Firstly, input is prepared by employing token, positional, and temporal embedding on an encoder's original time series input sequence. The prepared data is provided to the ProbSpare self-attention block, which helps remove canonical self-attention and reduce the network size. Instead of using Eq. 7, ProbSpare self-attention uses Eq. 11.

$$Attention(Q, K, V) = softmax\left(\frac{\overline{Q}K^T}{\sqrt{d}}\right)V \tag{11}$$

Afterwards, the encoder layer is designed to extract reliable long-range dependencies from lengthy sequential inputs. Encoder layers add convolution, activation, and maximum pooling operations between each encoder and decoder layer and cut the input sequence of the preceding layer into half to eliminate the issue of too much memory utilization. Similarly, the decoder receives a long input sequence with 0 padding elements. The procedure follows Eq. 12.

$$X_{j+1}^t = MaxPool(ELU(Conv1d([X_j^t]_{AB}))) \tag{12}$$

Where $[X_j^t]_{AB}$ is the calculation result of the multi-headed ProbSparse self-attention layer in the previous layer; ELU is the activation function; X_{j+1}^t is the current layer output of the multi-headed ProbSparse self-attention layer.

In my previous research [24], we utilized Gaussian Mixture Model (GMM) clustering to identify heterogeneous machines within the data center. However, in this study, we opted for BIRCH clustering over GMM. This choice is rooted in BIRCH superiority in handling large datasets and its ability to perform incremental clustering. BIRCH efficiency in real-time applications, coupled with its capacity for iterative refinement of clusters, makes it a more suitable choice for the evolving and dynamic nature of data center environments, addressing scalability and

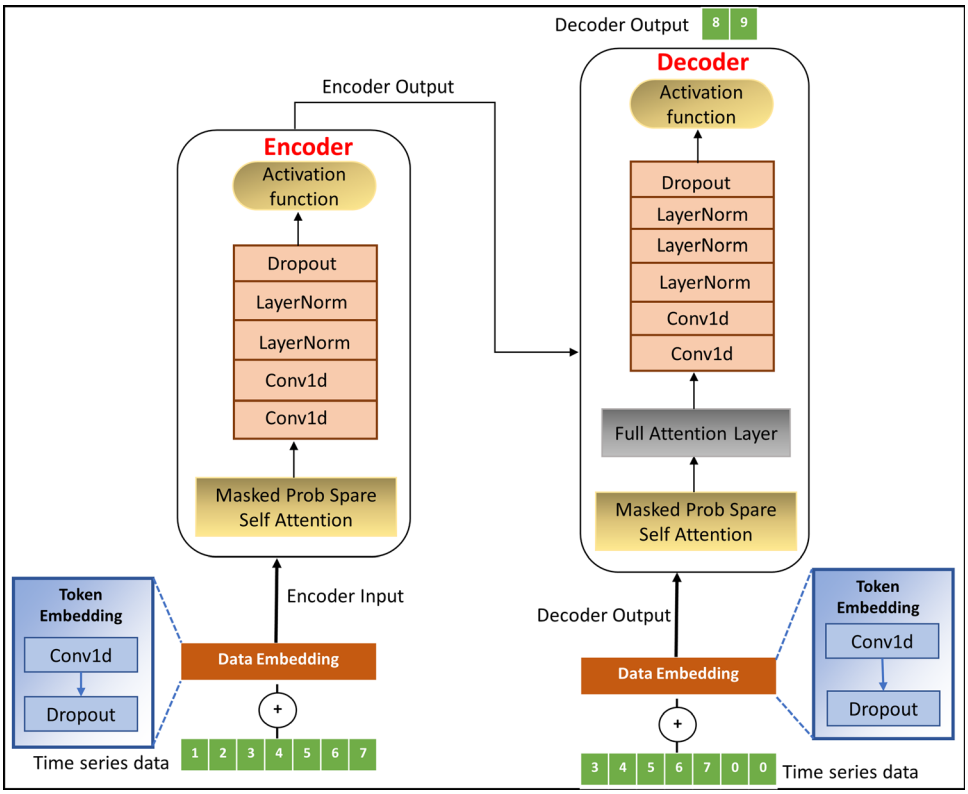


Fig. 3 Architecture of Informer model

processing efficiency concerns that may arise with GMM clustering.

4 Methodology

This section provides a detailed explanation of the methodology used to predict resource utilization on physical servers in a cloud data centre. As depicted in Fig. 4, the process proceeds in the following manner: Initially, the dataset is pre-processed to meet the clustering requirements, facilitating the extraction of heterogeneous physical machines based on their mean resource usage. Afterwards, with the selected machines identified, the data is further processed in accordance with the time series models. Finally, we employed LSTM, Transformer, and Informer models to predict the CPU usage of the machine. The primary objective of this methodology is to identify the most suitable models that can accurately predict CPU usage with minimal error, regardless of the configuration of cloud machines.

The methodology is divided into three subsections: Data pre-processing, Selection of heterogeneous machines, and Model implementation.

4.1 Data pre-processing

4.1.1 Dataset: google cluster trace usage dataset

We evaluated the employed prediction models using Google Cluster Trace Usage (GCT) dataset. This public dataset contains anonymized job and task data from Google's production cluster [34]. The resource usage table of dataset includes information on the resource usage of individual tasks within jobs, where each row in the table corresponds to a single task and contains the following information:

- Start time: The time at which the task started executing, measured in microseconds.
- End time: The time at which the task finished executing, measured in microseconds.
- Job ID: Unique id for each user's arriving request.
- Task index: A task identifier with respective to job Id.
- Machine ID: A unique identifier for the machine on which the task was running.
- Mean CPU usage: The average amount of CPU time used by the task, measured in CPU seconds.
- Canonical memory usage: The average amount of memory used by the task, measured in bytes.
- Assigned memory usage: Number of pages accessed by users.
- Mean local disk space used: The amount of disk space used by the task, measured in bytes.
- Mean disk I/O time: Mean across all disks on the machine, in units of disk-time seconds per second.
- Memory accesses per instruction: Measurement of last-level cache misses is used to estimate memory accesses.
- Sample portion: The ratio of the expected sample size to the observed sample size
- Total page cache memory usage: Total Linux page cache
- Unmapped page cache memory usage: Page cache not mapped into any userspace process
- Maximum disk usage: The maximum amount of disk space used by the task at any point during its execution, measured in bytes.

The feature selection process was driven by the specific requirements of our study, which involves clustering

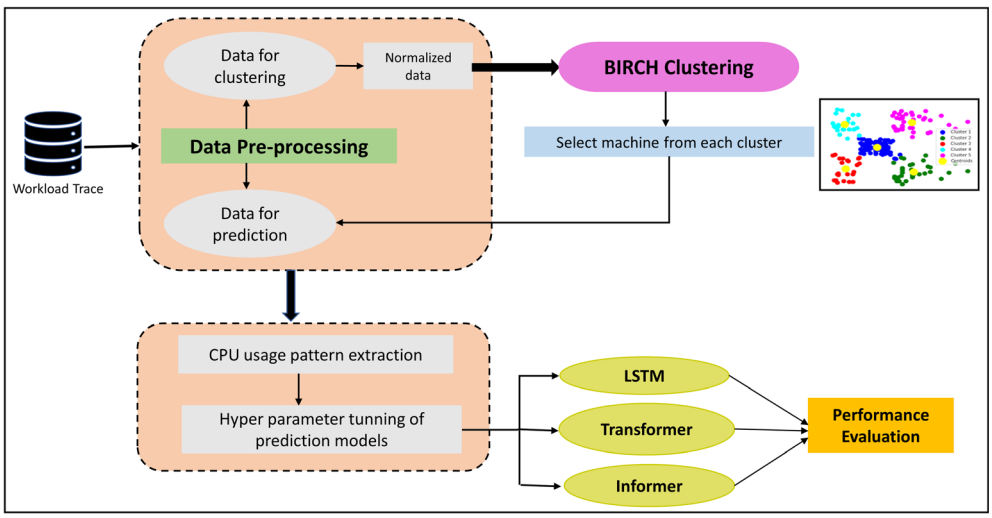


Fig. 4 Methodology to predict CPU usage of physical machines

machines based on mean resource usage and mean CPU usage prediction. The GCT dataset encompasses attributes associated with resource categories, including CPU, memory, and hard disk specifications.

To initiate the process, we first extracted a subset of heterogeneous machines from the dataset. Heterogeneity was defined based on the average utilization of resources by these machines. To facilitate machine classification into clusters, we selected six features from the dataset: Start time, End time, Machine ID, mean CPU usage, canonical memory usage, and mean local hard disk space usage. The inclusion of Start time and End time is crucial for calculating 5-min time intervals, as detailed in Sect. 4.1.2. Machine IDs were incorporated to provide unique identifiers, streamlining the grouping of machines efficiently.The features mean CPU usage, canonical memory usage, and mean local hard disk space usage are fundamental for computing the total average utilization of the corresponding resources. Only these features are required to serve as the basis for dividing machines into clusters, enabling the selection of a representative set of heterogeneous machines.

Subsequently, for predicting the mean CPU usage of physical machines, we further refined our feature selection. For this objective we focused on three key features: Start time, End time, and mean CPU usage. The 5-min time intervals were computed using Start time and End time, while mean CPU usage was utilized to train predictive models. In pursuing this objective, we exclusively focus on predicting mean CPU usage. Hence, our feature selection is centered solely around the inclusion of the mean CPU usage feature.

Regarding the remaining features in the dataset, we intentionally omitted certain attributes that were not directly aligned with our study's objectives. Our focus on the selected six features ensures that we prioritize the most relevant information for clustering machines based on mean resource usage and three features for predicting mean CPU usage.

4.1.2 Pre-processing

Data pre-processing is the primary step towards the accurate resource prediction of a physical machine. Adequate pre-processing helps to clean the data and keep it consistent and meaningful. The resource usage table consists of task resource usage information instead of physical machine information, but the notion of research is to predict CPU usage of a physical machine. We also observed that resources are shared between multiple tasks in the same period.

Therefore, we calculate the mean resource usage of each physical machine available in the dataset for clustering and calculate the mean CPU usage of the physical machine over a 5-min interval for prediction. First, extract the dataset's start time, end time, canonical memory usage, mean CPU usage, and mean disk usage features for pre-processing and employ the Sum-Average (SA) [24]. Figure 5 depicts the procedure followed to calculate the mean CPU usage of a machine over 5 min (300 s) using the Sum-Average algorithm. Figure 5a shows six tasks running over a single machine. Here, $task_1$, $task_2$, and $task_6$, consume 30%, 50%, and 40% CPU, respectively, for 50 sec at the different time stamp. CPU usage of $task_3$ is 60%, and $task_4$ is 33% for 70 sec and 80 sec, respectively. $task_5$ consumes 50% of the CPU for 100 sec to complete it. Figure 5b shows how the SA algorithm divides time intervals based on sharing of CPU resources between multiple tasks. The benefit of division is calculating the intact value of total mean CPU consumption. Here, $task_1$ uses the CPU for 30 sec individually out of 50 sec, and the rest of 20 sec is shared with $task_2$. Hence, total CPU consumption from 30 sec to 50 sec is 80%. Similarly, $task_3$ processes for 40 sec individually, i.e., from 120 sec to 150 sec, and the rest of the 30 sec CPU is shared with $task_4$. Similarly, $task_5$ and $task_6$ share the CPU from 230 sec to 280 sec. Lastly, after aggregating shared CPU usage, SA calculates the mean of total CPU utilization corresponding to the physical machine.

4.2 Selection of heterogeneous machines

The dataset comprises $12.5K$ heterogeneous machine information, so predicting resource usage for all available physical machines in the data center is computationally expensive and time-consuming. Thus, applying prediction models to every physical machine to check model performance is a cumbersome task for academia and researchers. This step also helps to train and evaluate prediction models' performance on complex heterogeneous patterns. Therefore, we use the clustering strategy and select a rationally set of heterogeneous configured physical machines from an entire dataset based on CPU, RAM, and hard disk utilization. Figure 6 depicts the procedure for selecting machine ids from the available physical machines.

Firstly extract start time, end time, machine id, canonical memory usage, mean CPU usage, and mean hard disk usage and employ the SA algorithm to find the mean resource usage of all the machines for 29 days. The processed data is normalized using "Z" normalization, which helps to ensure that all features are on the same scale by considering the mean of all data points as 0 and the standard deviation as 1 [55]. Afterwards, Hopkins statistics is employed on normalized data to check whether clustering is an effective approach for the data or not, using

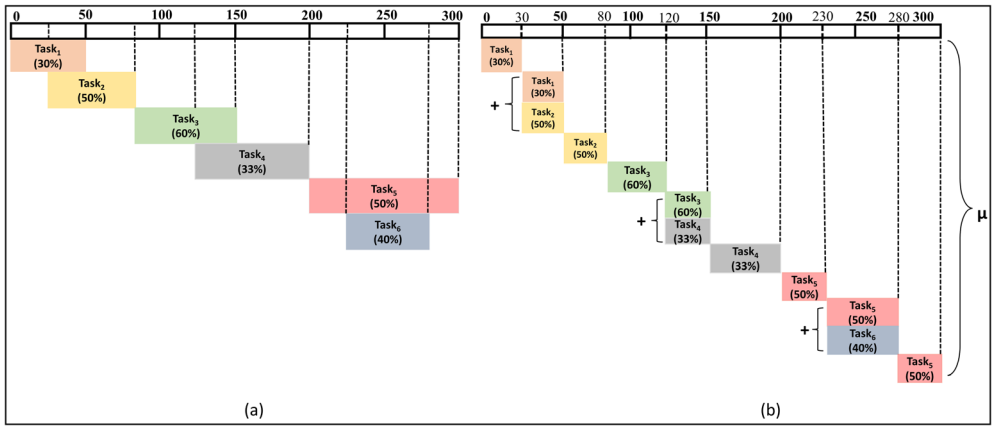


Fig. 5 **a** Utilization of CPU by a task. **b** Procedure use by Sum-Average algorithm

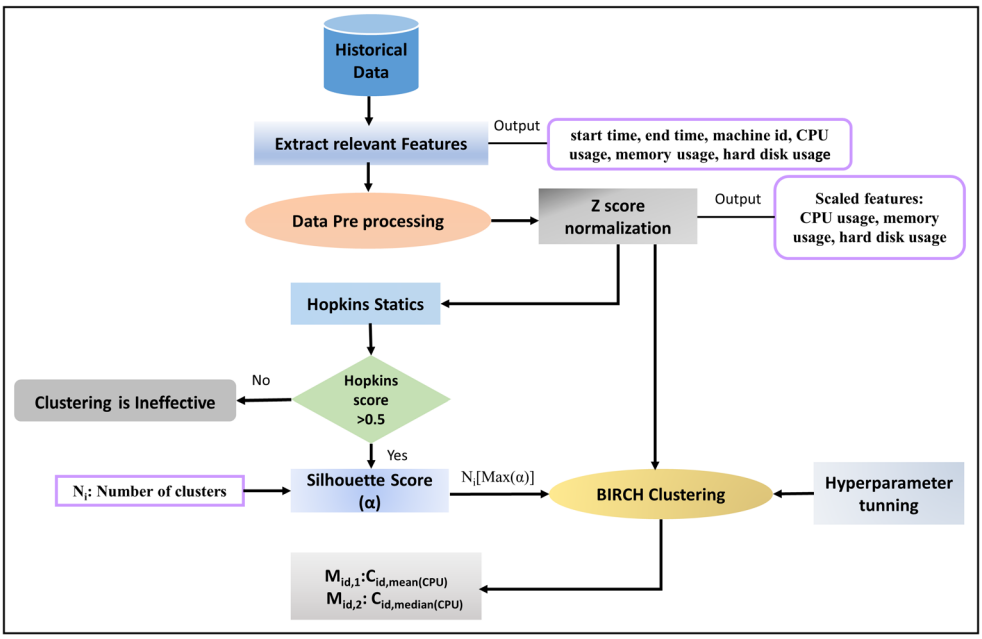


Fig. 6 Selection of heterogeneous machines

probability estimation by checking the clustering tendency. If Hopkins statistics $(H) > 0.5$, it means clustering is a practical approach; otherwise, the data does not qualify for clustering [56]. Hopkin's result is more significant than 0.5, suggesting that we can create groups to divide entire machines based on CPU, memory, and hard disk usage.

Finally, Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) clustering algorithm was employed to assign each available data centre machine to a specific cluster. Therefore, analyze the data using silhouette analysis to find the optimal number for clustering and select two machines from each cluster to represent those clusters [57]. Each cluster is represented by the machines that are closest to its mean and median of mean CPU usage. Further, the prediction of mean resource usage is conducted using these retrieved machine ids. As a result, we collected two physical machines from each cluster; therefore, we gathered the total number of physical machines is twice the number of optimal clusters.

Algorithm 1 demonstrates the procedure followed to divide all physical machines into clusters and which machines are selected for the prediction step.

Algorithm 1 clsMch: Machine selection to represent the entire available machines in a data center using clustering technique

Input: $[D] = \left(d_{pq}\right)_{p=1,q=1}^{p=k,q=l}$ Where, $d_{p,q} = q^{th}$ feature of p^{th} file, k = files in task resource table table, l= number of features in k.

Output: $ID_{\{c,1\}} = \left\{M_{\{C_p,mean\}}\right\}_{p=1}^{k}$, $ID_{\{c,2\}} = \left\{M_{\{C_p,median\}}\right\}_{p=1}^{k}$ Where, C_p = Cluster number, M = Machine Id, $mean$ = machine nearest to the mean of cluster, $median$ = machine nearest to the median of cluster, k = total number of clusters.

Initialize an empty matrix G and store features of time series data from D data set
$G = (y_1, y_2, y_3, y_4, y_5, y_6)$ where, y_1 = start_time, y_2 = end_time, y_3 = machine_id, y_4 = mean_CPU_usage, and y_5 = canonical_memory_usage, y_6 = mean_diskspace_used.
$Datapreprocessing(G) \rightarrow$matrix(T)
$(T_{ij} - \mu)\sigma \rightarrow T_{\text{new}}$ where, i-th row and j-th column
$Hopkins(T_{new}) \rightarrow H$
if $H > 0.5$ **then**
    $Silhouette\_score(T_{new}, n) \rightarrow S$ where, n = 4,5,6,7,8,9,10,11
    Find n corresponding to max(S) $\rightarrow N$
    $BIRCH(T, branching\_factor, N, threshold) \rightarrow C_k$ *where,* $k = 1$—N
    for j from 1 to N **do**
        Extract $C_j(T) \rightarrow F_j$
        Calculate $F_j(y_4).mean() \rightarrow \alpha_j$
        Find $F_j(y_3)$ whose $F_j(y_4)$ equal to α_j or nearest to α_j
        Calculate $F_j(y_4).median() \rightarrow \beta j$
        Find $F_j(y_3)$ whose $F_j(y_4)$ equal to βj or nearest to βj
    end for
else
    Clustering is ineffective approach for T data
end if

4.3 Model implementation

For implementation, firstly, extract the features from data F={start time, end time, and mean CPU usage} of collected heterogenous machines using clustering. Afterwards, the Sum Average algorithm is used to pre-process the data to calculate the mean CPU usage of physical machines over a 5 min interval. Lastly, the employed LSTM, Transformer and Informer are trained and tested with processed data and analyses their performance based on the root mean square error (RMSE) metric. This section is divided into two subsections: environment setup and model setup.

4.3.1 Environment setup

The proposed methodology uses the Python programming language (version 3.9.12) in the PyCharm Python Integrated Development Environment. The implementation is conducted on a computer system configured with a 12th Gen Intel(R) Core(TM) i7-12700 H 2.30 GHz, GPU Nvidia GeForce RTX 3060 with 16 GB of physical memory, and Microsoft Windows 10 Professional on a 64-bit operating system.

4.3.2 Model setup

The processed data of physical machines are divided into a 7:3 ratio, where 70% of the data is used to train models, and the rest 30% is used for testing the performance of applied models. We implemented each model over 2 (10 min), 6 (30 min), 12 (1 h), 18 (1.5 h), 24 (2 h), and 48 (4 h) input window sizes to investigate the effect of input sequence dependencies and compared their performance to determine the optimal size.

Further, to determine the optimal hyperparameters for each model, we conducted a randomized search over a predefined range of values for learning rate, batch size, epoch and the number of hidden units. We examine multiple combinations of hyperparameters on the test set of models and choose the configurations that produce the best results. In addition, we trained the Informer model on combinations of token, positional and temporal embedding

layers and compared their results to determine the optimal arrangement.

The significance of the proposed approach is to provide insight into the impact of input sequences and long-range dependencies on accurate prediction of CPU utilization, regardless of the machine's configurations. By implementing this methodology, our objective is to determine the most efficient model for accurately predicting CPU utilization across various machine configurations. This analysis enables us to assess the effectiveness of LSTM, Transformer, and Informer models in capturing the complexity of input sequences and long-term dependencies. Ultimately, it helps us identify the most suitable model for accurate predictions of CPU usage on diverse physical machines.

5 Results and discussion

5.1 Selection of heterogeneous machines

The achieved Hopkins statistic of 0.9549 on the normalized resource usage data confirms the possible grouping of physical machines. We calculate the silhouette score from k= 2 to 12 range to determine how closely a machine resembles its group compared to other clusters. According to silhouette statistics, the number of cluster corresponding to the highest score is considered an optimal number for clustering. Figure 7a depicts that a silhouette score of 0.61 is highest at $k = 5$. Therefore, we employed the BIRCH algorithm to divide the entire physical machines of the Google cluster into five groups based on CPU, memory, and hard disk usage. To divide the machines into clusters we used six features: start time, end time, machine ID, mean CPU usage, canonical memory usage and mean local disk space usage. Fig. 7b depicts that all available physical machines are seamlessly divided into five clusters using a 3D image where each colour denotes a specific cluster, and the axis indicates the resource's type.

Afterwards, we calculate each cluster's mean and median CPU usage and extract the machine id whose CPU usage is near the computed values. Lastly, we extracted ten machine ids, i.e., two machine id from each cluster for further analysis. Table 1 represents the selected machine ids corresponding to their cluster number. Here, machine 1 and machine 2 attributes denote machine id near to mean and median of CPU usage, respectively.

5.2 Analysis of Informer embedding layers

Firstly, we employed the Informer model with a token, positional, and temporal embedding layer to predict the mean CPU usage of physical machines and achieved high RMSE as shown in Fig. 8. Further, we tested the Informer model with a token and positional embedding layer and observed improved results, as the predicted value is too close to the actual value. At last, we employed the Informer model with a token embedding layer, and the results outperformed each selected machine. Therefore, we consider only the token embedding layer of the Informer model to predict mean CPU usage of physical machine. Figure 8 represents the achieved RMSE value of mean CPU usage prediction for machine id 4820094424 after testing with 3 (token, positional, and temporal), 2 (token and positional), and 1 (token) embedding layers over 2, 6, 12, 18, 24, and 48 input window sizes.

Table 1 Selected machine ids corresponding to cluster id

Cluster Id	Machine 1	Machine 2
1	4820094424	257335556
2	38708463	306881505
3	1676250332	1338630
4	351653579	3365958041
5	1335688	317486393

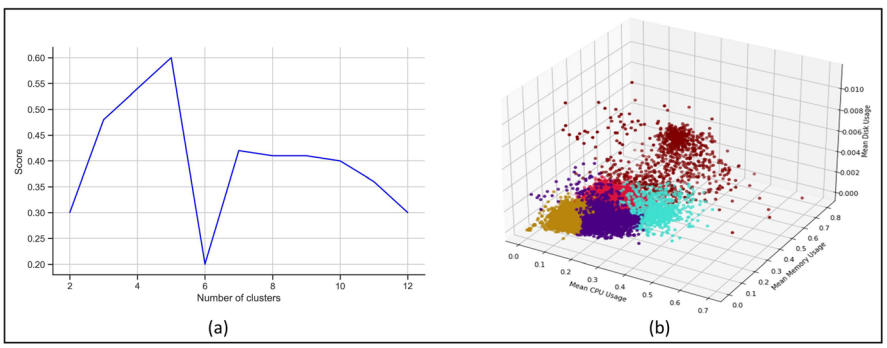


Fig. 7 **a** Number of clusters vs Silhouette score. **b** Clusters formed using BIRCH clustering technique

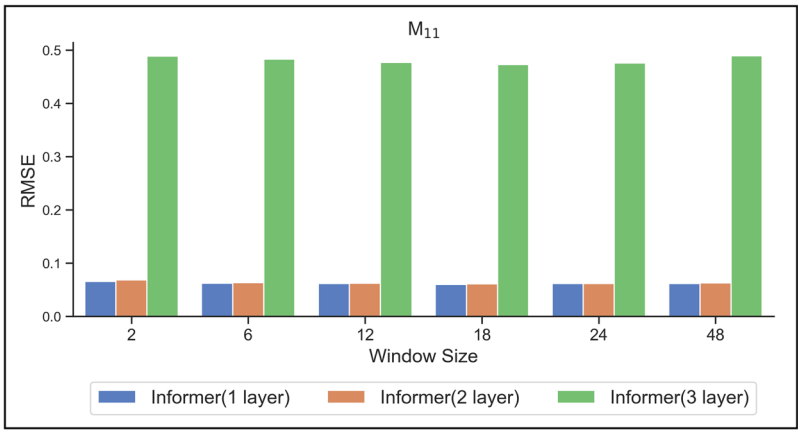


Fig. 8 RMSE comparison between token, positional, and temporal embedding layer of Informer

Table 2 LSTM hyperparameters

Epoch	80	Batch size	64
Hidden layers	30	Optimizer	Adam
Activation function	Relu	Dropout	0.01

Table 3 Transformer hyperparameters

Epoch	80	Batch size	64
Feature size	250	Num layer	1
Dropout	0.01	Nhead	10
Max length	5000	Learning rate	0.0001

Table 4 Informer hyperparameters

Epoch	10	Batch size	64
Time features encoding	Minute	Features	S
Encoder layer	1	Decoder layer	1
Dmodel	256	Nhead	8
Device	GPU	Dropout	0.05
Learning rate	0.0001	Attention	Prob
Embedding layer	Token embedding	Patience	5
Sequence length	2/6/12/18/24/48		
Label length	2/6/12/18/24/48		

5.3 Optimal hyperparameter of models

We experimented with various hyperparameters for LSTM, Transformer and Informer to optimize its performance for CPU usage prediction of physical machines. After conducting multiple trials, we identified the following hyperparameter values that resulted in the best performance. The optimal hyperparameters for LSTM, Transformer and Informer are described in Tables 2, 3, and 4, respectively.

5.4 Performance comparison of implemented models

The achieved RMSE values for each model during mean CPU usage prediction were plotted to compare their performances. Here, M_{ij} denotes j^{th} machine of i^{th} cluster where, $i \in \{1, 2, 3, 4, 5\}$ and $j \in \{1, 2\}$.

Figure 9 shows the achieved RMSE of LSTM, Transformer and Informer over 2,6,12,18,24, and 48 input window sizes for selected machines of cluster 1. Figure 9a depicts that for machine 4820094424, LSTM and Transformer achieved the lowest RMSE score of 0.068562 and 0.066298 at window size 12, whereas, Informer got RMSE of 0.06185. The Informer performs best at window size 18 with an RMSE of 0.060492. Figure 9b represents the RMSE difference between Transformer and Informer with respect to the LSTM model in percentages ($\Delta RMSE_{Model-LSTM}$) for machine 4820094424. The result shows that at window size 2, the change in RMSE of the Transformer is almost negligible. It shows that Transformer and Informer reduced RMSE at each input window size whereas, at window size 18, Informer performs best by reducing RMSE 14% compared to LSTM. Figure 9c depicts the results of machine id – 257335556. Here, LSTM and Transformer achieved the lowest RMSE at window size 12, with 0.065014 and 0.064987, respectively, and Informer achieved 0.064635. The Informer performs better at window size 18 with an RMSE of 0.06416. Figure 9d shows that for machine 257335556, the RMSE difference between Transformer and LSTM is small at window sizes 12 and 18. The Informer and Transformer reduce RMSE for each window size, whereas, at window size 48, they reduced the most by 5.5% and 2.8% compared to LSTM. Figure 10a, b shows the achieved pattern of

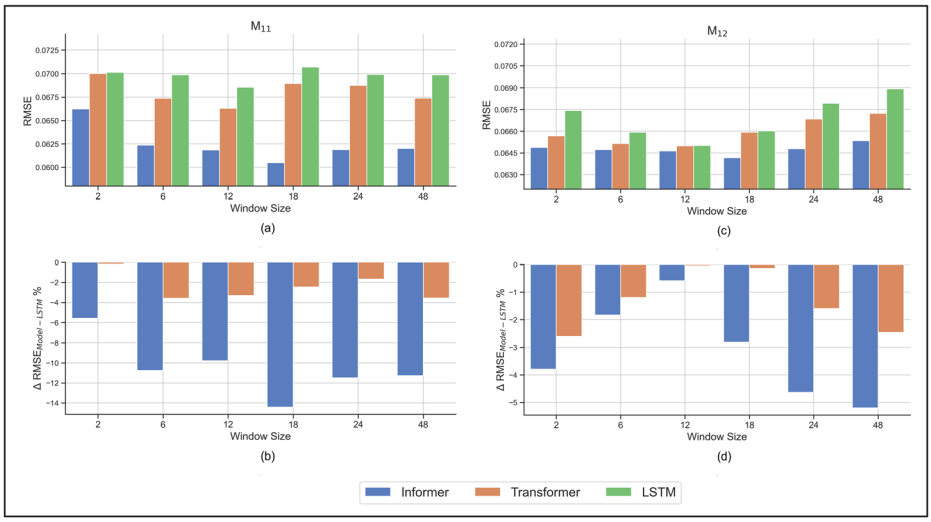


Fig. 9 Comparison between the models using RMSE for different window sizes. **a**, **c** RMSE for machines 4820094424 and 257335556; **b**, **d** Change in RMSE of Informer and Transformer with respect to LSTM for machines 4820094424 and 257335556

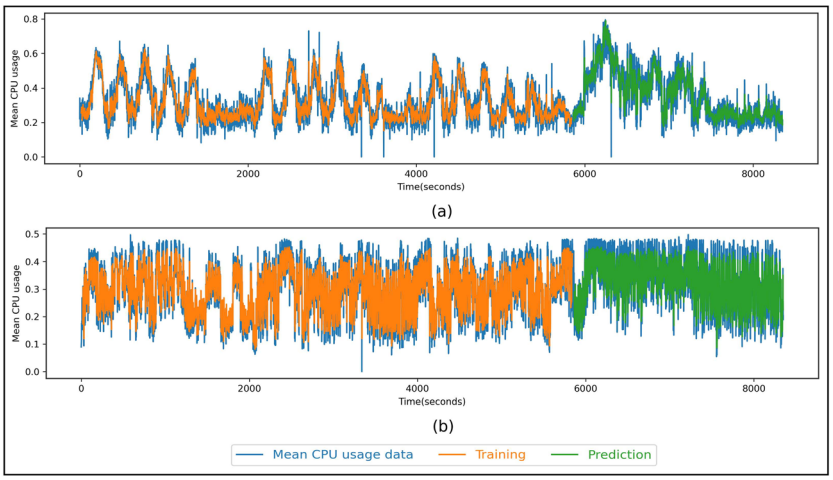


Fig. 10 Actual and predicted mean CPU usage by Informer at window size = 18. **a** Machine 4820094424, **b** Machine 257335556

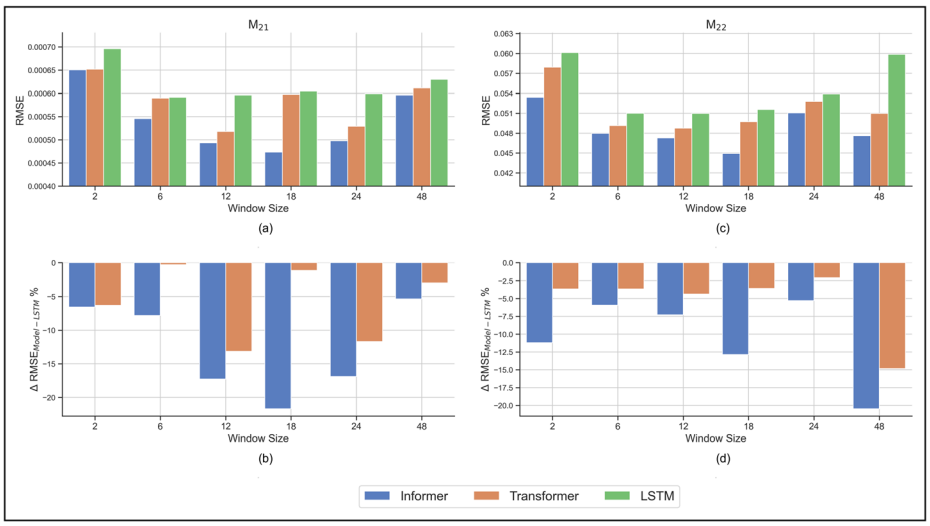


Fig. 11 Comparison between the models using RMSE for different window sizes. **a**, **c** RMSE for machines 38708463 and 306881505; **b**, **d** Change in RMSE of Informer and Transformer with respect to LSTM for machines 38708463 and 306881505

training and predicting mean CPU usage by Informer over actual observation at window size 18 for machines 4820094424 and 257335556, respectively.

Figure 11 represents the RMSE of employed models over 2,6,12,18,24 and 48 input window sizes for selected machines of cluster 2. Figure 11a shows the results of

machine 38708463; LSTM achieved the lowest RMSE of 0.000592 at window size 6, whereas Transformer performed best at window size 12 with an RMSE of 0.000518. The Informer outperforms with an RMSE of 0.000473 at window size 18. It is observed that the RMSE of Informer decreases as the window size increases to 18 and then keeps increasing. Figure 11b shows that at every window size, Informer and Transformer reduced RMSE compared to the LSTM model for machine 38708463, and Informer performed best by lowering the RMSE by 28%. However, at window sizes 6 and 18, the RMSE difference between Transformer and LSTM is small. Figure 11c depicts the achieved RMSE of machine 306881505, where LSTM and Transformer perform best at window size 12 with an RMSE of 0.050983 and 0.048754, respectively, and the Informer model gets 0.047271. At window size 18 Informer model outperform with an RMSE of 0.04493. Figure 11d shows that the RMSE of Informer and Transformer model is less than the LSTM model at each window size for machine 306881505. The Informer model decreases RMSE by 22% at window size 48.

Figure 12a, b displays the actual, trained and predicted mean CPU usage patterns achieved by Informer at window size 18 for machines 38708463 and 306881505, respectively. Figure 13 represents the RMSE of applied models over input window sizes of 2, 6, 12, 18, 24, and 48 for selected machines of cluster 3. Figure 13a represents LSTM performs best at window size 12 with an RMSE of 0.06593, and Transformer model performs best at window size 6 with an RMSE of 0.06419 for machine 1676250332. Whereas, Informer outperforms other models on window size 18 with an RMSE of 0.06300. Figure 13b shows that Transformer and Informer reduced RMSE at every input window size compared to the LSTM. Transformer and Informer deduced RMSE by 5.7% and 6.8%, respectively. At window size 12, the difference of Transformer and LSTM is negligible. Figure 13c depicts that LSTM performs best at window size 6 and Transformer at window size 12 with RMSE of 0.05316 and 0.05189, respectively, for machine 1338630. Here Informer performs best compared to other employed models at window 18 with an RMSE of 0.04959. Figure 13d shows that Transformer achieved a 5.3% deduction at window size 12, and Informer achieved a 9.8% deduction at window size 18 in RMSE of machine 1338630 compared to the LSTM. But at window sizes 2 and 6, Transformer model RMSE is almost equivalent to LSTM.

Figure 14a, b depicts that the prediction pattern of mean CPU usage is close to actual mean CPU usage for machines 1676250332 and 1338630, respectively, by employing the Informer model at window size 18. Figure 15 represents the RMSE of applied models over input window sizes of 2, 6, 12, 18, 24, and 48 for selected machines of cluster 4. Figure 15a represents that LSTM achieved the best mean CPU usage prediction at window size 12, and Transformer performed best at window size 6 with an RMSE of 0.06523 and 0.06378, respectively, for machine 351653579. The Informer outperformed at window size of 18 with achieved an RMSE of 0.06286. Figure 15b shows that the Transformer and Informer outperform at each window size of machine 351653579. Whereas, Transformer reduced RMSE by 3.5% at window size 6 and Informer reduced by 5.9% at window size 24 compared to the LSTM. At window size 12, Transformer is achieving almost similar RMSE as LSTM. Figure 15c depicts the results corresponding to machine 3365958041 and represents that the LSTM, Transformer and Informer achieved the best performance at window size 18 with an RMSE of 0.054399, 0.053734, and 0.052976, respectively. Figure 15d represents that at window size 12, the Informer reduced the RMSE by 3.9% compared to LSTM for machine 3365958041. However, at window sizes 2 and 6, the difference between Transformer and LSTM RMSE is small.

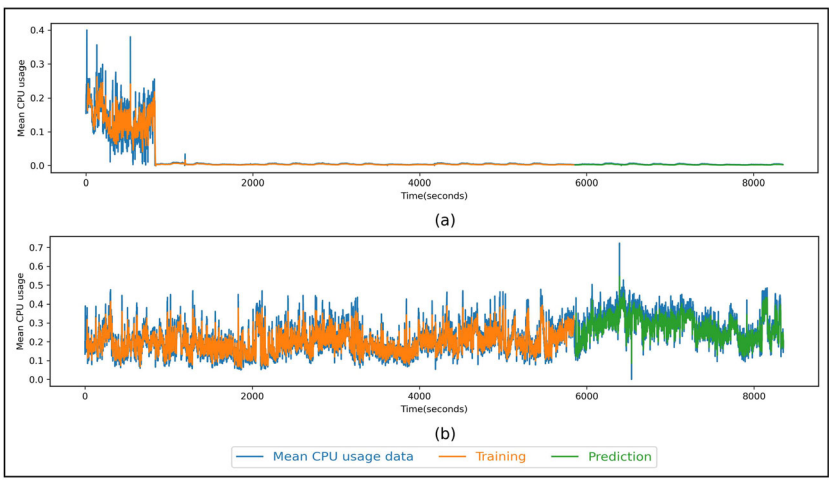


Fig. 12 Actual and predicted mean CPU usage by Informer model at window size= 18. **a** Machine 38708463, **b** Machine 306881505

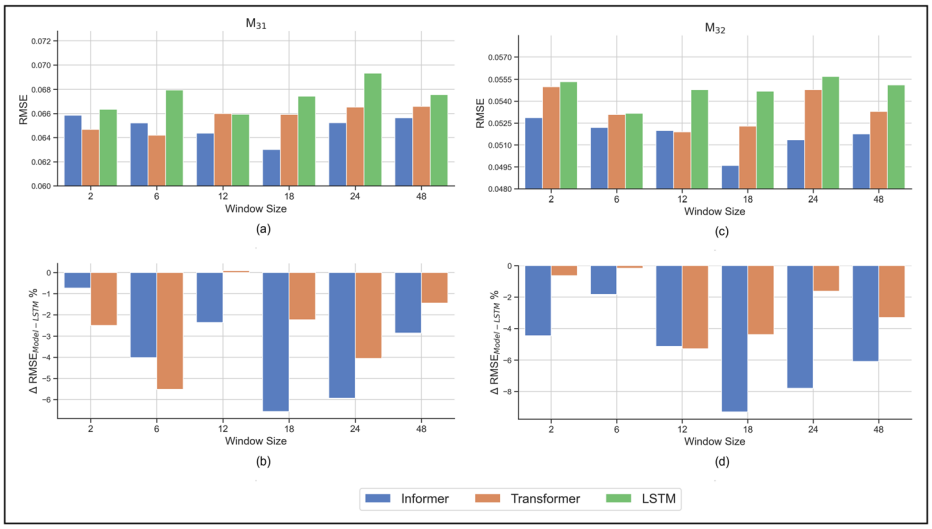


Fig. 13 Comparison between the models using RMSE for different window sizes. **a**, **c** RMSE for machines 1676250332 and 1338630; **b**, **d** Change in RMSE of Informer and Transformer with respect to LSTM for machines 1676250332 and 1338630

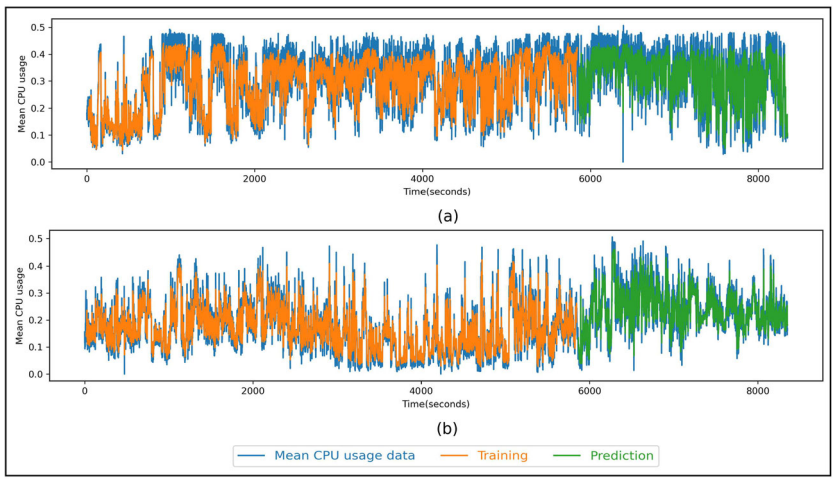


Fig. 14 Actual and predicted mean CPU usage by Informer model at window size= 18. **a** Machine 1676250332, **b** Machine 1338630

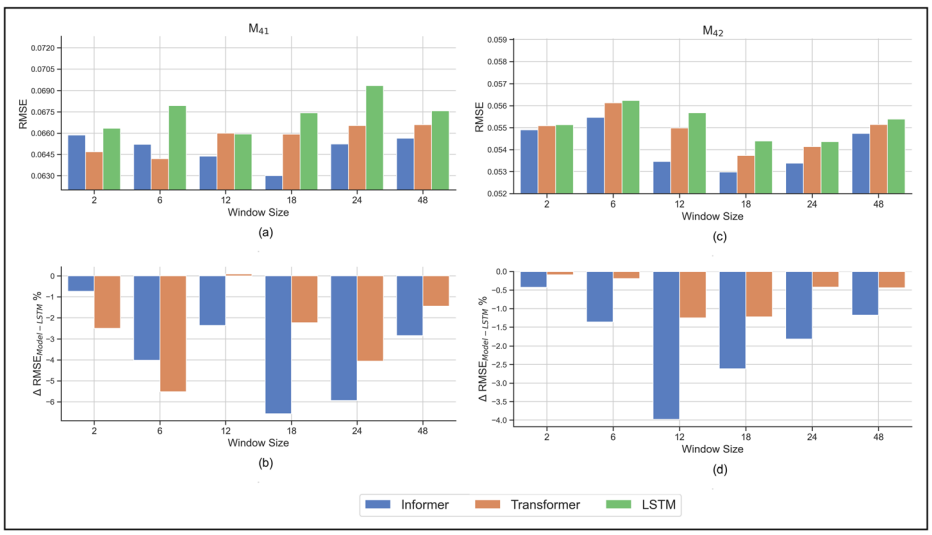


Fig. 15 Comparison between the models using RMSE for different window sizes. **a**, **c** RMSE for machines 351653579 and 3365958041; **b**, **d** Change in RMSE of Informer and Transformer with respect to LSTM for machines 351653579 and 3365958041

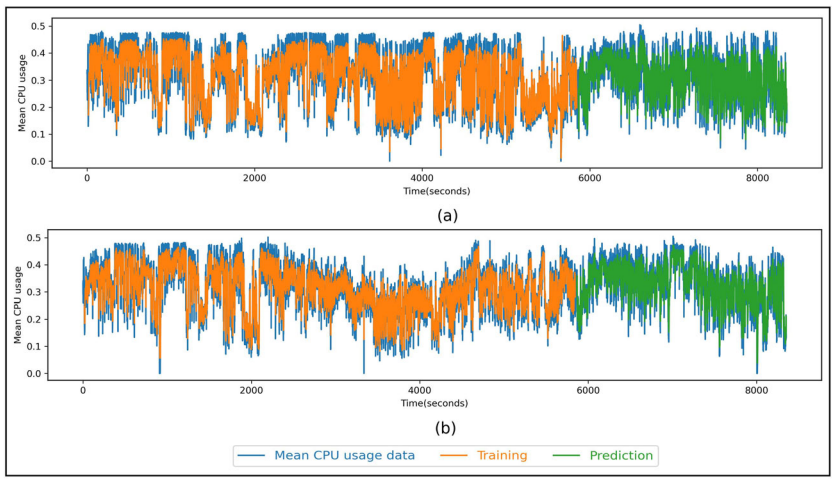


Fig. 16 Actual and predicted mean CPU usage by Informer model at window size= 18. **a** Machine 351653579, **b** Machine 3365958041

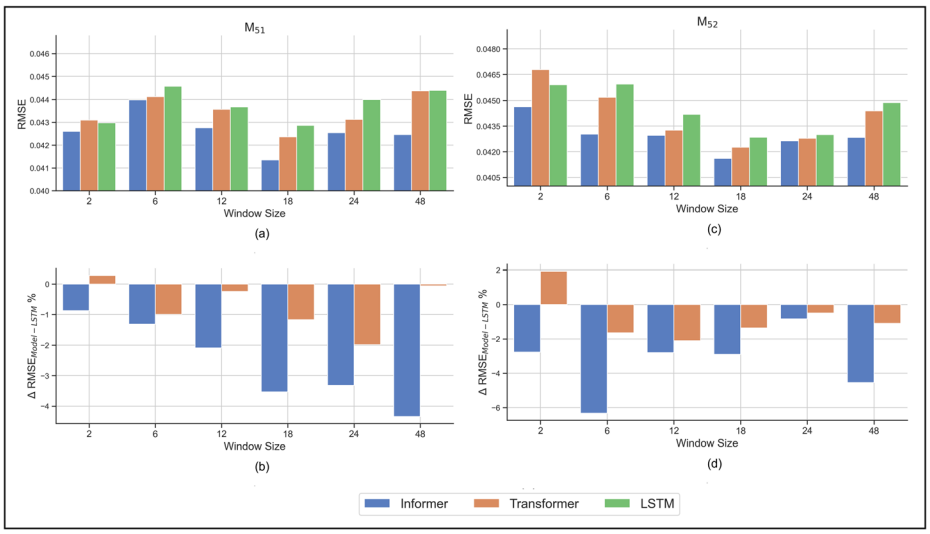


Fig. 17 Comparison between the models using RMSE for different window sizes. **a**, **c** RMSE for machines 1335688 and 317486393; **b**, **d** Change in RMSE of Informer and Transformer with respect to LSTM for machines 1335688 and 317486393

Figure 16a, b depicts the trained and predicted CPU usage pattern by employing the Informer at window size 18 for machine 351653579 and 3365958041, respectively.

Figure 17 represents the RMSE of applied models over input window sizes of 2, 6, 12, 18, 24, and 48 for selected machines of cluster 5.

Figure 17a shows LSTM, Transformer, and Informer performed best at a window size 18 with an RMSE of 0.04286, 0.04236, and 0.04135, respectively, for machine 1335688; hence, Informer outperforms. Figure 17b denotes that for machine 1335688, at window size 2, LSTM performs better than Transformer. Further, at window size 48, the RMSE of Transformer and LSTM are almost similar, and at window size 12, the RMSE difference between Transformer and LSTM is low. Whereas, Informer outperforms at each window size and reduces RMSE by 4.8% compared to LSTM at window size 48. Figure 17c depicts that prediction performance enhances correspondingly to the increment of window size up to 18 for machine 317486393. The best achieved RMSE of LSTM at window size 18 and Transformer at window size 12 is 0.04285 and 0.04325, respectively. The Informer outperforms with an RMSE of 0.04161 at window size 18. Figure 17d shows that for machine 317486393, at window size 2, LSTM beat than Transformer and at window size 24, the RMSE difference between Transformer and LSTM is slight. However, Transformer reduces RMSE by 2.1% at window size 12, and Informer by 6.2% at window size 6 compared to LSTM.

Figure 18a, b depicts the trained and predicted mean CPU usage value by employing the Informer at an input window size 18 for machines 1335688 and 317486393, respectively.

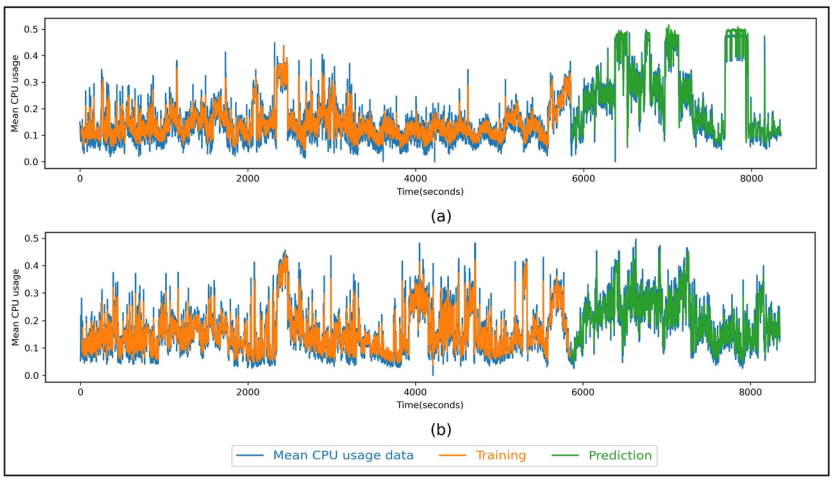


Fig. 18 Actual and predicted mean CPU usage by Informer model at window size= 18. **a** Machine 1335688, **b** Machine 317486393

6 Conclusion and future work

Based on the results of our studies, we can support the hypothesis that consider both long-range dependencies and input sequence order play a crucial role in time-series resource usage prediction in a cloud environment. We employed and evaluated three deep learning models: LSTM, Transformer, and Informer, to predict the CPU usage of physical machines. LSTM considers the sequence order of the input data when making predictions. Specifically, the model processes the input data sequentially, one element at a time, and updates its internal state at each step. Transformer worked well on long-range dependencies and processed the input data in parallel without considering the order of the input sequence. Whereas, Informer utilized both features.

The experiment results showed that the Transformer model predicts better than LSTM for most of the machines. This indicates that long-range dependencies significantly improve the prediction. For some machines, LSTM displayed better RMSE than Transformer, which concluded that the input sequence order is also relevant for resource usage predictions.This is highlighted by the better performance of the Informer model in all cases considering both aspects.

This study is limited to the use of CPU usage as the variable for prediction. As future investigations, the study could benefit from incorporating additional variables, such as memory and hard disk usage. The inclusion of these variables has the potential to contribute to the development of a more comprehensive and multivariate predictive model. Further, it would be advantageous to assess the predictive capabilities of the models with more datasets.

Funding No funding involved for this research.

Data availability The Google cluster trace usage dataset is publicly available at https://github.com/google/clusterdata/blob/master/ClusterData2019

Declarations

Conflict of interest The authors have not disclosed any competing interests.

References

1. Abid, A., Manzoor, M.F., Farooq, M.S., Farooq, U., Hussain, M.: Challenges and issues of resource allocation techniques in cloud computing. KSII Trans. Internet Inf. Syst. (TIIS) **14**(7), 2815–2839 (2020). https://doi.org/10.3837/tiis.2020.07.005
2. Madni, S.H.H., Latiff, M.S.A., Coulibaly, Y., Abdulhamid, S.M.: Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. Clust. Comput. **20**, 2489–2533 (2017). https://doi.org/10.1007/s10586-016-0684-4
3. Hameed, A., Khoshkbarforoushha, A., Ranjan, R., Jayaraman, P.P., Kolodziej, J., Balaji, P., Zeadally, S., Malluhi, Q.M., Tziritas, N., Vishnu, A., et al.: A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing **98**, 751–774 (2016). https://doi.org/10.1007/s00607-014-0407-8
4. Ahuja, R., Mohanty, S.K.: A scalable attribute-based access control scheme with flexible delegation cum sharing of access privileges for cloud storage. IEEE Trans. Cloud Comput. **8**(1), 32–44 (2017)
5. Calheiros, R.N., Masoumi, E., Ranjan, R., Buyya, R.: Workload prediction using Arima model and its impact on cloud applications' GOS. IEEE Trans. Cloud Comput. **3**(4), 449–458 (2014). https://doi.org/10.1109/TCC.2014.2350475
6. Gao, J., Wang, H., Shen, H.: Machine learning based workload prediction in cloud computing. In: 2020 29th International Conference on Computer Communications and Networks (ICCCN), pp. 1–9 (2020). https://doi.org/10.1109/ICCCN49398.2020.9209730 . IEEE
7. Chen, J., Wang, Y.: A hybrid method for short-term host utilization prediction in cloud computing. J. Electr. Comput. Eng. (2019). https://doi.org/10.1155/2019/2782349
8. Anupama, K., Shivakumar, B., Nagaraja, R.: Resource utilization prediction in cloud computing using hybrid model. Int. J. Adv.

Comput. Sci. Appl. (2021). https://doi.org/10.14569/IJACSA.2021.0120447
9. Dabral, P., Murry, M.Z.: Modelling and forecasting of rainfall time series using Sarima. Environ. Proc. **4**(2), 399–419 (2017). https://doi.org/10.1007/s40710-017-0226-y
10. Arora, P., Mehta, R., Ahuja, R.: An adaptive medical image registration using hybridization of teaching learning-based optimization with affine and speeded up robust features with projective transformation. Clust. Comput. (2023). https://doi.org/10.1007/s10586-023-03974-3
11. Adamuthe, A.C., Gage, R.A., Thampi, G.T.: Forecasting cloud computing using double exponential smoothing methods. In: 2015 International Conference on Advanced Computing and Communication Systems, pp. 1–5 (2015). https://doi.org/10.1109/ICACCS.2015.7324108 . IEEE
12. Ren, X., Lin, R., Zou, H.: A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, pp. 220–224 (2011). IEEE
13. Huang, J., Li, C., Yu, J.: Resource prediction based on double exponential smoothing in cloud computing. In: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 2056–2060 (2012). IEEE
14. Rahman, Z.U., Hussain, O.K., Hussain, F.K.: Time series qos forecasting for management of cloud services. In: 2014 Ninth International Conference on Broadband and Wireless Computing, Communication and Applications, pp. 183–190 (2014). https://doi.org/10.1109/BWCCA.2014.144 . IEEE
15. Chandy, A., et al.: Smart resource usage prediction using cloud computing for massive data processing systems. J. Inf. Technol. **1**(02), 108–118 (2019). https://doi.org/10.36548/jitdw.2019.2.006
16. Deepika, T., Prakash, P.: Power consumption prediction in cloud data center using machine learning. Int. J. Electr. Comput. Eng. (IJECE) **10**(2), 1524–1532 (2020). https://doi.org/10.11591/ijece.v10i2.pp1524-1532
17. Bankole, A.A., Ajila, S.A.: Predicting cloud resource provisioning using machine learning techniques. In: 2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1–4 (2013). IEEE
18. Mehmood, T., Latif, S., Malik, S.: Prediction of cloud computing resource utilization. In: 2018 15th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT), pp. 38–42 (2018). IEEE
19. Duggan, M., Mason, K., Duggan, J., Howley, E., Barrett, E.: Predicting host cpu utilization in cloud computing using recurrent neural networks. In: 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 67–72 (2017). IEEE
20. Borkowski, M., Schulte, S., Hochreiner, C.: Predicting cloud resource utilization. In: Proceedings of the 9th International Conference on Utility and Cloud Computing, pp. 37–42 (2016). https://doi.org/10.1145/2996890.2996907
21. Mason, K., Duggan, M., Barrett, E., Duggan, J., Howley, E.: Predicting host CPU utilization in the cloud using evolutionary neural networks. Futur. Gener. Comput. Syst. **86**, 162–173 (2018). https://doi.org/10.1016/j.future.2018.03.040
22. Lin, S.-Y., Chiang, C.-C., Li, J.-B., Hung, Z.-S., Chao, K.-M.: Dynamic fine-tuning stacked auto-encoder neural network for weather forecast. Futur. Gener. Comput. Syst. **89**, 446–454 (2018). https://doi.org/10.1016/j.future.2018.06.052
23. Shen, H., Hong, X.: Host load prediction with bi-directional long short-term memory in cloud computing. arXiv preprint arXiv:2007.15582 (2020). https://doi.org/10.48550/arXiv.2007.15582
24. Garg, S., Ahuja, R., Singh, R., Perl, I.: Gmm-lstm: a component driven resource utilization prediction model leveraging lstm and gaussian mixture model. Clust. Comput. (2022). https://doi.org/10.1007/s10586-022-03747-4
25. Ouhame, S., Hadi, Y., Ullah, A.: An efficient forecasting approach for resource utilization in cloud data center using CNN-lstm model. Neural Comput. Appl. **33**, 10043–10055 (2021). https://doi.org/10.1007/s00521-021-05770-9
26. Song, X., Liu, Y., Xue, L., Wang, J., Zhang, J., Wang, J., Jiang, L., Cheng, Z.: Time-series well performance prediction based on long short-term memory (lstm) neural network model. J. Petrol. Sci. Eng. **186**, 106682 (2020). https://doi.org/10.1016/j.petrol.2019.106682
27. Torres, J.F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., Troncoso, A.: Deep learning for time series forecasting: a survey. Big Data **9**(1), 3–21 (2021). https://doi.org/10.1089/big.2020.0159
28. Parmezan, A.R.S., Souza, V.M., Batista, G.E.: Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. Inf. Sci. **484**, 302–337 (2019). https://doi.org/10.1016/j.ins.2019.01.076
29. Wu, N., Green, B., Ben, X., O'Banion, S.: Deep transformer models for time series forecasting: The influenza prevalence case. arXiv preprint arXiv:2001.08317 (2020)
30. Yu, Y., Si, X., Hu, C., Zhang, J.: A review of recurrent neural networks: Lstm cells and network architectures. Neural Comput. **31**(7), 1235–1270 (2019). https://doi.org/10.1162/neco_a_01199
31. Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W.: Informer: Beyond efficient transformer for long sequence time-series forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 11106–11115 (2021). https://doi.org/10.1609/aaai.v35i12.17325
32. Farahnakian, F., Ashraf, A., Pahikkala, T., Liljeberg, P., Plosila, J., Porres, I., Tenhunen, H.: Using ant colony system to consolidate VMS for green cloud computing. IEEE Trans. Serv. Comput. **8**(2), 187–198 (2014)
33. Farahnakian, F., Pahikkala, T., Liljeberg, P., Plosila, J., Hieu, N.T., Tenhunen, H.: Energy-aware VM consolidation in cloud data centers using utilization prediction model. IEEE Trans. Cloud Comput. **7**(2), 524–536 (2016). https://doi.org/10.1109/TCC.2016.2617374
34. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format+ schema. Google Inc., White Paper **1**, 1–14 (2011)
35. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: a new data clustering algorithm and its applications. Data Min. Knowl. Disc. **1**, 141–182 (1997)
36. Xu, M., Song, C., Wu, H., Gill, S.S., Ye, K., Xu, C.: ESDNN: deep neural network based multivariate workload prediction in cloud computing environments. ACM Trans. Internet Technol. (TOIT) (2022). https://doi.org/10.1145/3524114
37. Mrhari, A., Hadi, Y.: Workload prediction using VMD and TCN in cloud computing. J. Adv. Inf. Technol. (2022). https://doi.org/10.12720/jait.13.3.284-289
38. Leka, H.L., Fengli, Z., Kenea, A.T., Tegene, A.T., Atandoh, P., Hundera, N.W.: A hybrid cnn-lstm model for virtual machine workload forecasting in cloud data center. In: 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), pp. 474–478 (2021). https://doi.org/10.1109/ICCWAMTIP53232.2021.9674067 . IEEE
39. Dang-Quang, N.-M., Yoo, M.: An efficient multivariate autoscaling framework using bi-lstm for cloud computing. Appl. Sci. **12**(7), 3523 (2022). https://doi.org/10.3390/app12073523
40. Karim, M.E., Maswood, M.M.S., Das, S., Alharbi, A.G.: Bhyprec: a novel bi-lstm based hybrid recurrent neural network model to predict the cpu workload of cloud virtual machine.

IEEE Access **9**, 131476–131495 (2021). https://doi.org/10.1109/ACCESS.2021.3113714
41. Zhu, Y., Zhang, W., Chen, Y., Gao, H.: A novel approach to workload prediction using attention-based lstm encoder-decoder network in cloud environment. EURASIP J. Wirel. Commun. Netw. **2019**(1), 1–18 (2019). https://doi.org/10.1186/s13638-019-1605-z
42. Nguyen, H.M., Kalra, G., Kim, D.: Host load prediction in cloud computing using long short-term memory encoder-decoder. J. Supercomput. **75**(11), 7592–7605 (2019). https://doi.org/10.1007/s11227-019-02967-7
43. Patel, E., Kushwaha, D.S.: A hybrid cnn-lstm model for predicting server load in cloud computing. J. Supercomput. **78**(8), 1–30 (2022). https://doi.org/10.1007/s11227-021-04234-0
44. Nadendla, H.: Why are LSTMs struggling to matchup with Transformers? https://medium.com/analytics-vidhya/why-are-lstms-struggling-to-matchup-with-transformers-a1cc5b2557e3
45. Yang, Z., Liu, L., Li, N., Tian, J.: Time series forecasting of motor bearing vibration based on informer. Sensors **22**(15), 5858 (2022). https://doi.org/10.3390/s22155858
46. Zeng, A., Chen, M., Zhang, L., Xu, Q.: Are transformers effective for time series forecasting? arXiv preprint arXiv:2205.13504 (2022). https://doi.org/10.1609/aaai.v37i9.26317
47. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Adv. Neural Inf. Proc. Syst.**30**, (2017)
48. Tang, B., Matteson, D.S.: Probabilistic transformer for time series analysis. Adv. Neural. Inf. Process. Syst. **34**, 23592–23608 (2021)
49. Wu, S., Xiao, X., Ding, Q., Zhao, P., Wei, Y., Huang, J.: Adversarial sparse transformer for time series forecasting. Adv. Neural. Inf. Process. Syst. **33**, 17105–17115 (2020)
50. Mohammadi Farsani, R., Pazouki, E.: A transformer self-attention model for time series forecasting. J. Electrical Comput. Eng. Innov. (JECEI) **9**(1), 1–10 (2020). https://doi.org/10.22061/jecei.2020.7426.391
51. Qian, Y., Tian, L., Zhai, B., Zhang, S., Wu, R.: Informer-WGAN: high missing rate time series imputation based on adversarial training and a self-attention mechanism. Algorithms **15**(7), 252 (2022). https://doi.org/10.3390/a15070252
52. Guo, L., Li, R., Jiang, B.: A data-driven long time-series electrical line trip fault prediction method using an improved stacked-informer network. Sensors **21**(13), 4466 (2021). https://doi.org/10.3390/s21134466
53. Ryan, T.: LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras. https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2
54. Pranav, P.: Recurrent Neural Networks, the Vanishing Gradient Problem, and Long Short-Term Memory. https://medium.com/@pranavp802/recurrent-neural-networks-the-vanishing-gradient-problem-and-lstms-3ac0ad8aff10
55. Patro, S., Sahu, K.K.: Normalization: A preprocessing stage. arXiv preprint arXiv:1503.06462 (2015)
56. Lachlan, R., Verhagen, L., Peters, S., Cate, C.T.: Are there species-universal categories in bird song phonology and syntax? a comparative study of chaffinches (fringilla coelebs), zebra finches (taenopygia guttata), and swamp sparrows (melospiza georgiana). J. Comp. Psychol. **124**(1), 92 (2010). https://doi.org/10.1037/a0016996
57. Zhou, H.B., Gao, J.T.: Automatic method for determining cluster number based on silhouette coefficient. Adv. Mater. Res. **951**, 227–230 (2014). https://doi.org/10.4028/www.scientific.net/AMR.951.227

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

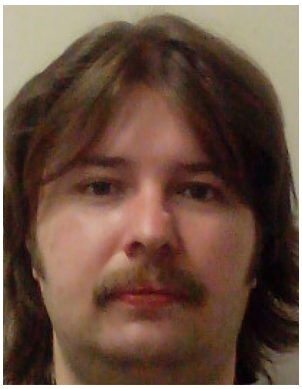
Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Sheetal Garg is pursuing a Ph.D. from Thapar Institute of Engineering and Technology, Patiala, Punjab, India. She received the B. Tech degree in Computer Science and Engineering from Graphic Era University, Dehradun, India in 2015, and the Master of Engineering in Computer Science from Thapar University, Patiala, Punjab, India in 2019. She worked as a software engineer in Infosys Pvt ltd, Chennai in 2016. Her primary research interest is in the area of Machine Learning, Deep Learning, and Cloud Computing. She has working experience in Big Data, .Net, C++, and Python programming language.

Rohit Ahuja has obtained his PhD degree in computer science and engineering discipline from the Indian Institute of Information Technology, Design and Manufacturing (IIITDM) Jabalpur India, he is currently working as an Assistant Professor with the Department of Computer Science and Engineering in Thapar Institute of Engineering and Technology (TIET) Patiala, India. He has supervised about 50 Undergraduate students and is currently supervising 4 Doctorate and guided 1 Master's students. He was a visiting researcher with Sakurai Lab, Kyushu University, Fukuoka, Japan. His research area is Artificial Intelligence and its applications.

Raman Singh received his Master of Engineering in information technology in 2010 and a Ph.D. degree in computer science & engineering in 2016 from Panjab University, Chandigarh India. He is an IEEE member since 2012. He is working as a Lecturer in the School of Computing, Engineering, and Physical Sciences, the University of the West of Scotland, Lanarkshire, the United Kingdom since November 2021. He has worked as a Post-Doctoral Fellow at the School of Computer Science and Statistics, Trinity College Dublin, The University of Dublin, Ireland from February 2020 to February 2021. He also worked as an Assistant

Professor in the Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala (India) from June 2016 to November 2021. His primary research interest is in the area of Machine Learning and Cyber and Network Security. He has completed Post-Doc Fellowship in Blockchain and Applied Cryptography and a Ph.D. in Machine learning-enabled Intrusion Detection systems. He has working experience in the following technologies: Machine Learning, ns-3 simulator, ndnSIM, Dockers Containers, Python programming language, Blockchain technology: Multichain and Ethereum, Matlab, and C++.



Ivan Perl Ph.D. in Computer Science, Associate Professor in the Department of Faculty - Informatics and Applied Mathematics, ITMO University. He graduated from Saint-Petersburg University of Fine Mechanics and Optics, now known as ITMO University (2009 Master's degree, 2012 - Ph.D.). For more than ten years has been working in the software development industry in Motorola and later in Oracle. Recent four years of engineering career were dedicated to design and development of high-scale enterprise solutions related to the Internet of Things (IoT). Research interests: high-efficient remote Earth sensing (RES), efficient cloud-based system dynamics modeling, mathematical modeling, computation networks, software engineering, and computer systems.