Future Generation Computer Systems 142 (2023) 376–392

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

MAG-D: A multivariate attention network based approach for cloud workload forecasting

Yashwant Singh Patel[1], Jatin Bedi[*,1]

Department of Computer Science Engineering, Thapar Institute of Engineering and Technology, Patiala, Punjab, India

ARTICLE INFO

Article history:
Received 13 May 2022
Received in revised form 19 December 2022
Accepted 2 January 2023
Available online 10 January 2023

Keywords:
Resources' utilization
Deep learning
Prediction approaches
Cloud data centers
Energy-efficiency
Time-series

ABSTRACT

The Coronavirus pandemic and the work-from-home have drastically changed the working style and forced us to rapidly shift towards cloud-based platforms & services for seamless functioning. The pandemic has accelerated a permanent shift in cloud migration. It is estimated that over 95% of digital workloads will reside in cloud-native platforms. Real-time workload forecasting and efficient resource management are two critical challenges for cloud service providers. As cloud workloads are highly volatile and chaotic due to their time-varying nature; thus classical machine learning-based prediction models failed to acquire accurate forecasting. Recent advances in deep learning have gained massive popularity in forecasting highly nonlinear cloud workloads; however, they failed to achieve excellent forecasting outcomes. Consequently, demands for designing more accurate forecasting algorithms exist. Therefore, in this work, we propose 'MAG-D', a **M**ultivariate **A**ttention and **G**ated recurrent unit based **D**eep learning approach for Cloud workload forecasting in data centers. We performed an extensive set of experiments on the Google cluster traces, and we confirm that MAG-DL exploits the long-range nonlinear dependencies of cloud workload and improves the prediction accuracy on average compared to the recent techniques applying hybrid methods using Long Short Term Memory Network (LSTM), Convolutional Neural Network (CNN), Gated Recurrent Units (GRU), and Bidirectional Long Short Term Memory Network (BiLSTM).

1. Introduction

During the COVID-19 pandemic, many industries and organizations are moving towards digitalization and automation. Cloud computing had a pivotal role in this transformation as it empowers remote work infrastructure with greater computation, agility, and storage capabilities [1,2]. To fulfill the diverse demands of cloud users, the cloud service providers (CSPs) enable virtualization technologies in the underlying physical infrastructure, which permits cloud applications to programmatically deliver networking & infrastructure-as-a-service (IaaS) [3]. To meet the service level agreements (SLA) requirements, all commercial CSPs primarily invest in designing optimal resource allocation strategies in such a way that the overall quality of experience (QoE) and quality of service (QoS) is enhanced [4]. A good workload forecasting approach plays a pivotal role in making informed decisions while meeting these constraints. Therefore, a number of key cloud players are performing extensive research in the development of workload forecasting algorithms to improve the auto-scaling performance to support the fluctuating and enormous big data workloads as well as to achieve significant energy savings [2,5–7]. The outcome of such strategies provides a massive reduction in overall power cost and carbon emission, achieving the goal of green cloud computing as shown in Fig. 1. In general, the cloud is a promising technology offering advantages of on-demand resources, global coverage, and high application availability [1,8]. However, the workload modeling and characterization is extremely challenging in a highly dynamic cloud environment. The workload modeling helps to improve the interpretation of typical cloud workload patterns resulting more informed decisions for robust resource management [9]. Through the workload characterization, performance models can be developed to support research issues such as energy-efficiency and resource management and to answer some critical research questions, such as: how are the overall cloud data centers' usage levels affected by customer behavior? How cloud data centers' energy-efficiency can be improved while the maintaining the QoS levels? [9]. Intuitively, in the absence of forecasting techniques, it is impossible to estimate real-time resource usage of cloud workload traces [7]. Therefore, the principal objective of this study is to design a more robust cloud workload forecasting algorithm.

[*] Corresponding author.
E-mail address: jatin.bedi@thapar.edu (J. Bedi).
[1] Both the authors contributed equally to the manuscript.

https://doi.org/10.1016/j.future.2023.01.002
0167-739X/© 2023 Elsevier B.V. All rights reserved.

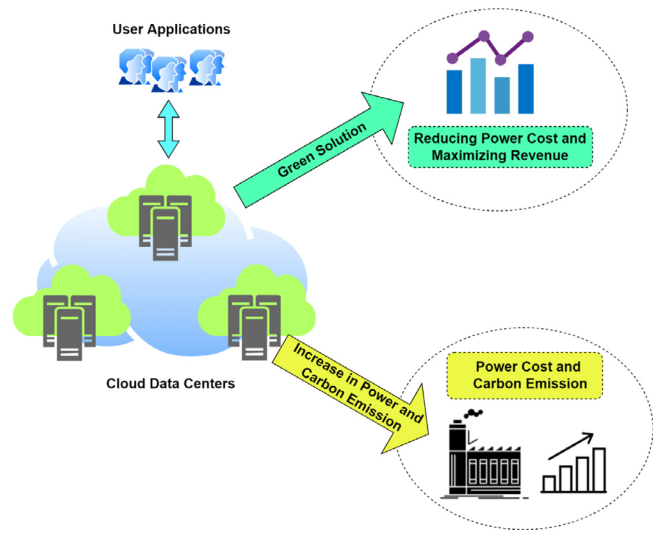

User Applications
Green Solution
Reducing Power Cost and Maximizing Revenue
Cloud Data Centers
Increase in Power and Carbon Emission
Power Cost and Carbon Emission

Fig. 1. Green cloud computing.

Over the past few decades, prediction approaches have received considerable attention in the domain of weather forecasting, stock market, and energy sectors [5,10,11]. For cloud workload prediction, various methods have been designed. These schemes can be arranged into statistical models, machine learning and deep learning-based methods. Statistical models of prediction presume linear dependency and stationary behavior between time-series samples. Most of the statistical techniques such as Holt-winter [12], ARIMA (Autoregressive integrated moving average) [13], seasonal ARIMA (SARIMA) [14], and Markov models [15,16] are extensively employed, but unfortunately, these approaches have dramatically failed to prove themselves in highly chaotic time series, and long-term forecasting applications [1]. On the other hand, the second category of machine learning techniques have been employed to address the shortcomings of these classical models [1]. Some of these methods, such as particle-swarm-optimization (PSO), support-vector-regression (SVR), and relevance-vector-machine (RVM), have been utilized to predict the workload in cloud data centers [1]. However, these methods are not suitable for large datasets, and their overall performance is highly dependent on tuning parameters. Deep learning-based prediction models are widely popular architecture performing superior than classical machine learning techniques to solve the most complex and challenging problems, such as cloud workload forecasting [8,17]. Convolutional neural networks (CNNs) extract input image features using convolution filters which can be used by the feature extraction network. The neural network utilized these extracted feature signals for classification [5]. To learn highly chaotic and volatile time series for detection of long-range dependence, LSTM (Long short-term memory) networks [18] have shown tremendous learning improvement [19]. Moreover, to improve the performance of forecasting, hybrid models are commonly designed [4,7]. It is an ensemble technique combining several models. While the methods applying empirical mode decomposition (EMD) are gaining significant attention which is a kind of nonlinear signal based-processing strategy and works constructively for frequency and time domains [4]. However, volatility is the key issue in the cloud workload forecasting. Thus, we require a robust nonlinear model to assess the dynamic changes and detect the long-range dependence in the cloud workload.

In cloud systems, different resource usage metrics are correlated. For example, we anticipate the CPU load to rise at the same pace to 40% and 60%, if it rises linearly from 0 to 20% over time. However, it is plausible to predict that the allocated memory will soon run out and start paging on disk, reducing CPU consumption in the future [20], given the rise in memory usage from 10% to 90% over the same time period. It demonstrates how the outcomes of resource consumption prediction are dependent on historical data (univariate time series forecasting) and the time series data of additional resource utilization-related factors (multivariate time series forecasting). These relationships highlight how crucial multivariate time series prediction is and make it possible to extract more features, which raises prediction performance. Therefore, it is crucial to investigate multivariate time series models instead of univariate forecasting models.

Along with the multivariate forecasting, it is essential to exploit the varying priority levels associated with a workload sequence at various points in time because of the different effects of earlier stages on the future host load. Thus, it is necessary to focus on the addition of an attention layer in deep learning networks. Also, most of the cloud workload prediction techniques [21–24] concentrate on single-step prediction. The one-step-ahead forecasting-based autoscaler uses inconsistent scale-up and scale-down actions [25] as a result of workload changes, wasting resources and expenses. To stop variations from impacting the scaling process, resource provisioning procedures must be carried out based on a multi-step-ahead forecasting.. To resolve the aforementioned issues, this paper aims to design a MAG-D architecture which applies multivariate attention layer and bidirectional GRU Network for multistep-ahead workload prediction. In fact, the main forecasting problem has been subdivided into machine learning (ML) pipeline including data preprocessing and data segmentation, clustering of time series inputs, stacked bidirectional GRU Network and bidirectional LSTM layer, and attention layer. Ultimately, we combine all independent results to get the final forecasted time step value. The novelty of this work is to propose a robust approach identifying the long-range dependence, effect of multiple resource usage metrics, and stacked deep learners for correct interpretation of data center workload patterns. The key contributions of the work are:

- Proposed a multivariate systematic deep learning architecture for cloud workload forecasting.
- Designed a MAG-D framework combining the attention layer & bidirectional GRU with improved prediction accuracy and reduced complexity.
- A clustering based stacked strategy is applied for improvement in the generalized performance.
- Using real cloud workload traces of Google cluster, we examine the performance in terms of RMSE & MAE.

The remaining of this work is structured as follows: Section 2 provides the related studies in the cloud literature of workload prediction. The basic principle and background study is introduced in Section 3. In Section 4, we present the complete architecture of proposed MAG-D model, and its complexity analysis. Implementation details and performance evaluations with real-world traces are discussed in Section 5. Lastly, Section 6 concludes the work.

2. Related work

For cloud workload forecasting, several studies are presented. These schemes can be categorized into statistical and ML based methods, deep learning based approaches, and hybrid schemes as depicted in Fig. 2.

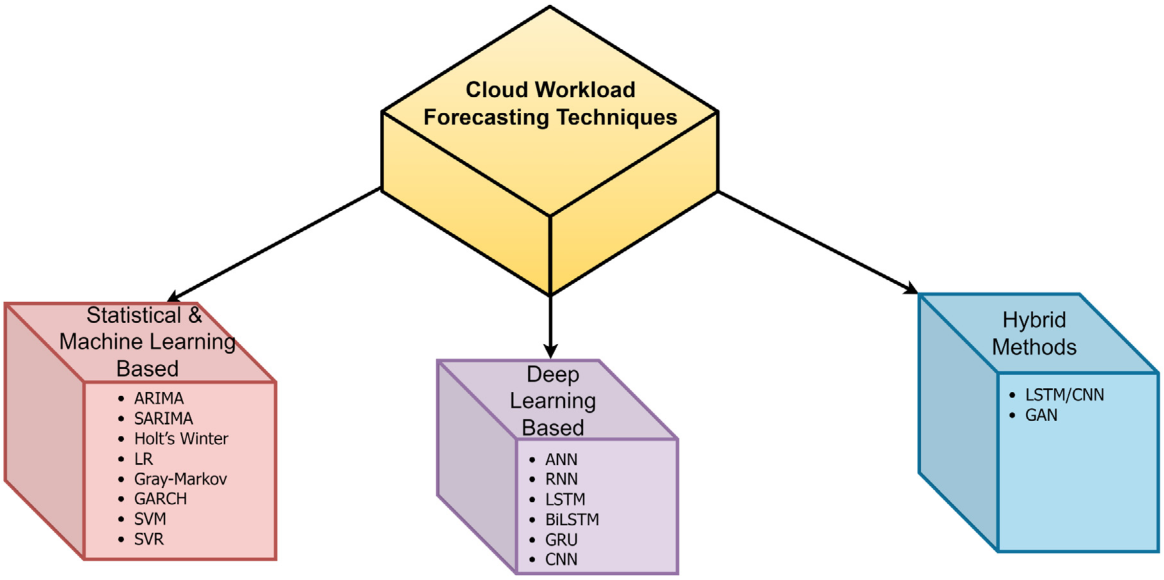


Fig. 2. Classification of cloud workload forecasting techniques.

2.1. Statistical and machine learning based cloud workload forecasting approaches

Some of the basic forecasting techniques namely Holt-winter (HW) [12], ARIMA [13], SARIMA [14], Markov models [15], and Feed-forward neural network [26] have been extensively applied in cloud workload forecasting. Zhang et al. [13] designed an ARIMA-enabled model to forecast cloud resource usage. ARIMA model is among one of the most popular time series forecasting models. Although, it does not performs well when nonlinearities exist in the time-series data. Hsieh et al. [27] proposed a Gray-Markov-based model for Virtual Machine (VM) consolidation, optimizing VM migrations, energy consumption, and SLA violations. Caglar et al. [26] designed an iOverbook model comprising a feed-forward mode of a 2-layer neural network, which generalizes the linear and nonlinear correlation of input & output by applying a single hidden layer. However, major of the studies have neglected the long-time period dependency of the resource usage metrics.

2.2. Deep learning based cloud workload forecasting approaches

Deep neural architectures have proved to be powerful and successfully applied for cloud workload forecasting in recent years. The most common architectures such as feed forward networks, recurrent neural networks (RNNs), gated recurrent units (GRUs), long-short term memory networks (LSTMs), bidirectional LSTM networks (BiLSTMs), and convolutional neural networks (CNNs) are gaining popularity for time-series forecasting [5,6,28,29]. Zhang et al. [2], proposed a self-adaptive differential evolution algorithm & neural network based workload forecasting strategy. For financial time-series forecasting, [30] designed a RNN approach. For cloud workload forecasting, a wavelet neural network based technique is presented in [31]. For a multi-input and single-output set of samples, polynomial neural network or Group meta-data handling (GMDH) approach builds a fusion of models in a self-organized form [4]. Such technique is utilized for container throughput prediction in [32]. Most of the approaches have subsumed that the time-series based data is either memoryless or stationary. In the series of cloud workload forecasting works, Ghorbani et al. [19] suggested the long-range dependence existence in the workload traces of Google's cluster [33], in which the past-time lags generally impacted the value of next time interval. In fact, the LSTM networks are said to be more efficient and robust to deal with such long-range dependence. Song et al. [34] suggested a univariate LSTM networks, which rigorously analyze the past CPU usage to forecast the future trends. In general, LSTMs are advanced RNNs, which works well on sequential sub-time series with long-term dependencies. In addition to univariate time series, [35,36], have observed that the multivariate approaches produces most promising results in comparison to univariate methods. Thus, this papers have applied all relevent features of Google's cluster workload traces such as CPU usage, maximum disk I/O time, aggregated page cache usage, unmapped amount of page cache, maximum CPU usages, per instruction memory accesses, disk I/O time and Cycles-Per-Instructions (CPIs), memory assigned, utilized memory, maximum memory usage observed, and space utilization of disk capacity etc. In addition to unidirectional LSTM, [37], Zhao et al. presented a bidirectional way of time-series network applying the extreme ML technique. Their model have produced high prediction accuracy. In the works by Gupta et al. [38], and [20], it is presented that the multivariate LSTMs and BiLSTMs networks generated highly accurate results on Google's cluster traces.

2.3. Hybrid methods for cloud workload forecasting

Ensemble learning has introduced the hybrid methods integrating the efficient deep learning models to detect some complex time-series prediction in different areas such as wind speed, energy load forecasting, traffic flow prediction, and financial time-series [5,10,28,39,40]. In the work of [41] a three level multi-model approach combining wavelet method, LSTM and ARIMA are proposed for forecasting with chaotic time series. Some other popular methods have combined the advantages of CNN and bidirectional stacked LSTM networks [40] for machine health monitoring. The authors in [42] proposed a hybrid approach integrating 1D ConvNets with stacked LSTM blocks in workload forecasting of Google data centers. Patel et al. proposed [43,44] multivariate forecasting-based hotspots and coldspots mitigation approach applying k means clustering with stacked BiLSTM networks for cloud workload forecasting. To explore the applications of generative adversarial networks (GANs) for time-series forecasting, Yazdanian et al. [45] proposed a multiscale ensemble model combining LSTM and GAN based deep learning architecture cloud workload forecasting, which has produced good prediction results with fluctuating time-series. Behera et al. [46] presented a multiscale deep bidirectional GRU based deep learning networks for remaining useful life (RUL) estimation on the C-MAPSS dataset. In comparison with LSTM of three gates, GRU applied two gates. This method produces superior learning performance than existing methods because of its simpler architecture and faster training yielding high prediction performance with highly fluctuating time-series and becomes one of the vertical of MAG-D architecture.

3. Problem definition and theoretical background

3.1. Problem definition

We have assumed the workload traces of a cloud data center having multiple resource usage metrics. In this context, multi-dimensional resource usage value T_{ru} and resource usage metric f (memory or CPU usage) are recorded at regular intervals from 1 to n. Consequently, we can denote an individual time instance entry of cloud resource usage as:

$$T_{ru} = \langle T^1_{ru},\ T^2_{ru},\ \dots\ T^n_{ru}\rangle \tag{1}$$

where each entry T^i_{ru} is a feature vector of the form $\langle T^{i1}_{ru},\ T^{i2}_{ru} \dots T^{if}_{ru}\rangle$. T^{if}_{ru} represent the target resource usage metric f at ith time step.

$$T^{n*f}_{ru} = \begin{bmatrix} T^{11}_{ru} & T^{12}_{ru} & .. & T^{1f}_{ru} \\ T^{21}_{ru} & T^{22}_{ru} & .. & T^{2f}_{ru} \\ T^{31}_{ru} & T^{32}_{ru} & .. & T^{3f}_{ru} \\ .. & .. & .. & .. \\ T^{n1}_{ru} & T^{n2}_{ru} & .. & T^{nf}_{ru} \end{bmatrix} \tag{2}$$

$T^f_{ru} = \langle T^{1f}_{ru},\ T^{2f}_{ru},\ \dots,\ T^{nf}_{ru}\rangle$ is a multivariate time-series for system's state by implying other resource metrics $T^{(n-1)*(f-1)}_{ru}$ with different time intervals.

We assume, training and testing resource usage set as H and J respectively. Now, the training and testing datasets of resource usage can be denoted as:

$$T^{train}_{ru} = T^{(H)*(f-1)}_{ru} \tag{3}$$

$$T^{test}_{ru} = T^{(J)*(f-1)}_{ru} \tag{4}$$

$$Z_{train} = T^{(H)*(f^{th})}_{ru} \tag{5}$$

$$Z_{test} = T^{(J)*(f^{th})}_{ru} \tag{6}$$

So, the aim of our research study is to solve the resource usage prediction problem by training a mapping function ($\mathcal{F}$) between the features set and the target usage values. Mathematically, the mapping function is given as:

$$\sum_{i=1}^{n} Z \implies \mathcal{F}(\sum_{i=1}^{n} T^{i*(f-1)}_{ru}) \tag{7}$$

The optimization function can be formulated as:

$$minimize \sqrt{\frac{1}{L}\sum_{i=1}^{L}(Z^i_{train} - \mathcal{F}(Z^i_{train}))^2} \tag{8}$$

where L denotes the total number of entries and Z^i_{train} represents the actual resource usage at ith time instance in the training dataset. The $\mathcal{F}(Z^i_{train})$ denotes the forecasted resource usage metric value using the mapping function $\mathcal{F}$

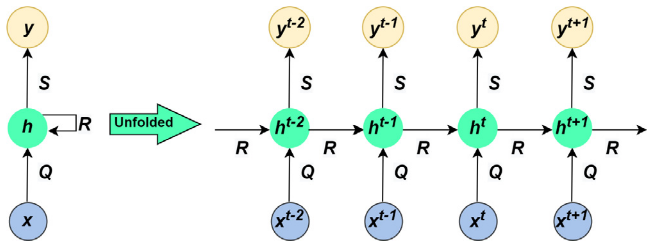


Fig. 3. Unfolded RNN.

3.2. Theoretical background

3.2.1. RNNs

A typical artificial neural network (ANN) [47] has three layers namely input, output, & hidden layers. Assuming input data denoted as x, and the output list and list of hidden states are represented by y, and h respectively. To map the input to hidden layer, a matrix Q is formed. To couple the hidden to hidden layer connections, matrix R is constructed. Another matrix S maps the output to hidden layer connections. But, the standard feed-forward neural networks are failed in the scenarios of obtaining patterns in input data over different timestamps. Whenever sequential inputs are involved, RNNs work efficiently. In general, RNNs [48] are deep neural networks comprising a self-connected hidden layer as shown in Fig. 3.

Let us assume a input sequence x for n time-steps data forming: $\{x^0, x^1, x^2, \cdot x^{n-1}\}$, hidden state $\mathbf{h}$, and output $\mathbf{y}$. At time-step t, we can express the hidden state h^t as:

$$h^t = \lambda(Rh^{t-1} + Qx^t + o^1) \tag{9}$$

The activation function is represented by λ and bias vector is denoted by o^1. In general, we can express the y^t as:

$$y^t = \lambda(Sh^t + o^2) \tag{10}$$

Here o^2 is a bias vector for y^t output.

For the non-linear activation function scenario, the sigmoid function can be formulated as:

$$\lambda(x) = \frac{1}{1+e^{-x}} \tag{11}$$

RNNs are based on the concept of back-propagation. However, one of the major limitation of simple RNN is "fading memory" due to the absence of standard structural formation. Due to this, a traditional RNN model recollect only the most up to date knowledge while neglecting the previous information.

3.2.2. LSTM networks

LSTMs [49] are mostly preferred to handle the long-range dependence issue, and performs fabulously on sequence-based tasks. The inclusion of hidden layer in the LSTM architecture makes it different from the RNN architecture. The hidden layer of LSTM model is also known as the LSTM cell. The fundamental LSTM block architecture is represented in Fig. 4.

LSTM architecture includes a memory cell S^t, an input gate I^t, an output gate G^t, and a forget gate F^t. The activation functions are Sigmoid and *tanh* denoted by σ and *tanh*, respectively. For point-wise multiplication, the $\otimes$ is used. The x^t expressed as the input at time instant t while the h^t is used for hidden state representation. To decide the amount of input information in cell state, K^t is used as the candidate state of memory cell. (R^F, Q^F) are used for the recurrent and input weight metrics where as o^F represent the bias for forget gate. Sigmoid function σ suggests which values are permitted to pass. The forget gate F^t is designed to validate

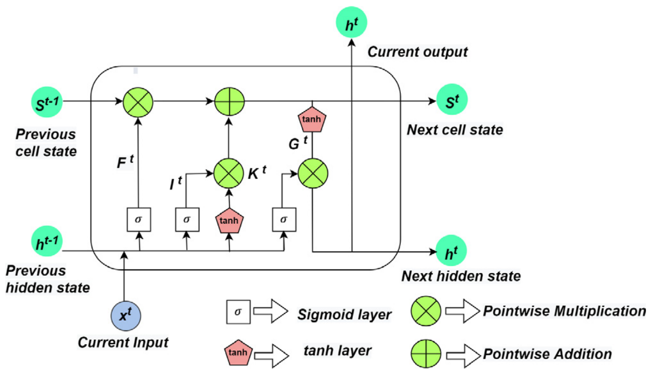


Fig. 4. LSTM.

the cell state contents. It decides which content requires to be discarded. It can be expressed as:

$$F^t = \sigma(x^t * Q^F + o^F + h_{t-1} * R^F) \tag{12}$$

Next, the cell state S^t is required to be updated with the new contents collected via the previous cell state S^{t-1}, input and forget gate.

$$S^t = S^{t-1} * F^t + K^t * I^t \tag{13}$$

A tanh layer is primarily selected for constructing the new candidate values K^t and formulated as:

$$K^t = tanh(o^K + x^t * Q^K + h^{t-1} * R^K) \tag{14}$$

$$I^t = \sigma(o^I + Q^I * x^{t-1} + R^I * h^{t-1}) \tag{15}$$

where (R^K, Q^K, o^K) and (R^I, Q^I, o^I) denote the weight matrices for recurrent and input, and bias vector for the cell state of candidate cell state and input gate correspondingly.

At-last, the output gate G^t is based on the cell state content. The sigmoid activation function mainly determines that which part of the fragment to be given as an output. It can be formulated as:

$$G^t = \sigma(o^G + x^t * R^G + h^{t-1} * R^G) \tag{16}$$

$$h_t = G^t * tanh(S^t) \tag{17}$$

In the formulation, (R^G, Q^G) denote the weight metrics for recurrent and input and o^G represents the output gate bias. In the figure, the symbol ⊗ denotes the element-wise multiplication.

3.2.3. Bi-directional LSTM

In comparison with unidirectional LSTM, the bidirectional networks produces superior performance in several applications. Bi-LSTMs [50] integrates the advantages of BiRNNs and LSTM networks so that it can execute the input sequence in both the directions. Fig. 5 represents the architecture of unfolded BiLSTM layer.

The $\overrightarrow{h}$ expresses the forward layer output time sequence, which is iteratively determined via the input sequence of time-series from $T - n$ to $T - 1$. Next, the $\overleftarrow{h}$ denotes the backward layer output sequence, which is concertedly computed using the reversed inputs sequence of time-series $T - 1$ to $T - n$. Applying classical LSTM equations from Eqs. (12) to (17), the outputs of forward and backward layer are obtained. Eventually, the BiLSTM layer generates the Y_T output vector. In this vector, each element is quantified by the following expression:

$$y_t = \sigma(\overrightarrow{h}, \overleftarrow{h}) \tag{18}$$

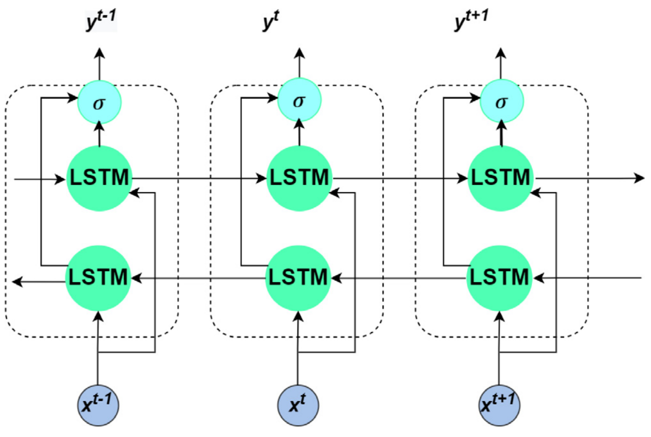


Fig. 5. Unfolded BiLSTM.

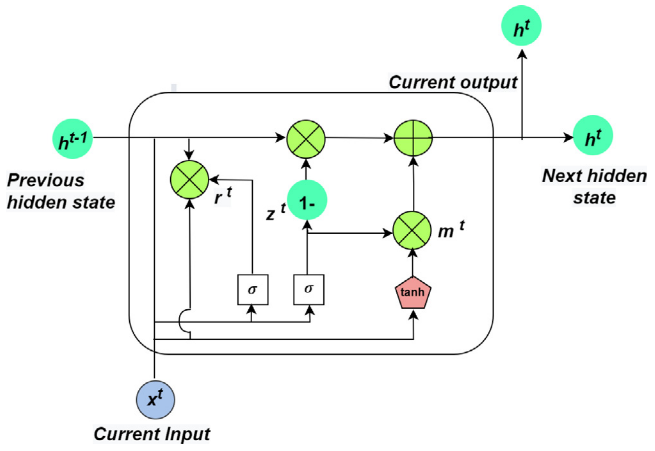


Fig. 6. GRU.

Here σ function merged the two output series. Next, the final BiLSTM layer output vector is formed as, $Y_T = [y_{T-n}, \dots, y_{T-1}]$. Here the y_{T-1} describes the forecasted value of next time step.

3.2.4. GRU

GRU [51] forms a simpler architecture and widely known as an alternative of LSTM. In the GRU architecture, the forget gate & input gate are integrated within one unit named as update gate utilizing only one hidden state. The fundamental structure of a classical GRU cell is represented in Fig. 6.

Assuming input weight matrix as Q^r, the recurrent weight matrix as R^r with bias as o^r at a particular time-step t. The reset gate g^t is allocated to embodied past memory along with new input. Once the r^t is closed, whole associated information is considered as unusable for the present hidden state and rejected.

$$g^t = \sigma(R^r h^{t-1} + Q^r x^t + o^r) \tag{19}$$

Considering input weight matrix Q^m, recurrent weight matrix R^m, and bias o^m, the candidate cell c^t is formulated using x^t, h^{t-1} and r^t as:

$$c^t = tanh(R^m((g^t \oplus h^{t-1} + Q^m x^t + o^m))) \tag{20}$$

While (R^z, Q^z) denotes the (recurrent and input) weight matrices and bias is represented as o^z, update gate as z^t, controls the flow on information via previous to present hidden state.

$$z^t = \sigma(R^z h^{t-1} + Q^z x^t + o^z) \tag{21}$$

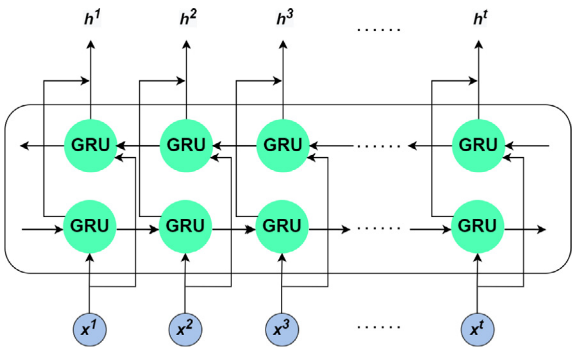


Fig. 7. BiGRU.

Now, we can obtain the final hidden state h^t through applying the addition operation of two composite expressions applying element-wise product as $\otimes$ for $(1-z^t)$, h^{t-1} and z^t, c^t correspondingly.

$$h^t = (1 - z^t) \otimes h^{t-1} + z^t \otimes c^t \tag{22}$$

The lesser gates are advantageous and GRU to form a light-weight and efficient architecture using lesser number of parameters producing faster training in comparison with LSTM network.

3.2.5. BiGRU

GRU is useful for modeling of temporal dependency in the sequential data across a longer period of time. Eventually, the overall execution get degraded due to the consideration of only past data while neglecting the future information. Bi-directional GRUs [52] are useful to resolve these issues, a fundamental model of BiGRU layer is shown in Fig. 7.

It is constructed using two stacked GRU layers: (i) first one flows in the forward direction computing forward hidden states as $\overrightarrow{h^1}, \overrightarrow{h^2}, \overrightarrow{h^3}, \ldots \overrightarrow{h^t}$, and provided input in forward direction from time-step 1 to t, and (ii) other one flows in backward direction computing the backward hidden states denoted via $\overleftarrow{h^1}, \overleftarrow{h^2}, \overleftarrow{h^3}$, $\cdot\,\overleftarrow{h^t}$, where the input represents the opposite direction for time-step t to 1, for any given c length of sequence. So that it can access the information of past as well as future. The equations from (23)–(26) represent the function of hidden layer for particular time-step t. To formulate the functions for backward direction $\leftarrow$, we utilized following equations:

$$\overleftarrow{g^t} = \sigma(\overleftarrow{R^r}\,\overleftarrow{h^{t+1}} + \overleftarrow{Q^r}\,\overleftarrow{x^t} + \overleftarrow{o^r}) \tag{23}$$

$$\overleftarrow{c^t} = tanh(\overleftarrow{R^m}((\overleftarrow{g^t} \oplus \overleftarrow{h^{t+1}} + \overleftarrow{Q^m}\,\overleftarrow{x^t} + \overleftarrow{o^m}))) \tag{24}$$

$$\overleftarrow{z^t} = \sigma(\overleftarrow{U^z}\,\overleftarrow{h^{t+1}} + \overleftarrow{P^z}\,\overleftarrow{x^t} + \overleftarrow{o^z}) \tag{25}$$

$$\overleftarrow{h^t} = (1 - \overleftarrow{z^t}) \oplus \overleftarrow{h^{t+1}} + \overleftarrow{z^t} \oplus \overleftarrow{c^t} \tag{26}$$

Here the input weight matrices for $\overleftarrow{x^t}$ input are denoted with $(\overleftarrow{Q^r}, \overleftarrow{Q^m}, \overleftarrow{Q^z})$ respectively.

For recurrent-weight matrices attached with hidden state $\overleftarrow{h^{t+1}}$, $\overleftarrow{R^r}$, $\overleftarrow{R^m}$, $\overleftarrow{R^z}$ are used respectively. To represent bias for the backward process, we used $\overleftarrow{o^r}$, $\overleftarrow{o^m}$, $\overleftarrow{o^z}$ respectively. In a BiGRU layer, the output of hidden state $\overleftarrow{h^t}$ can be estimated using the element-wise sum in both directions as follows:

$$h^t = \overrightarrow{h^t} \oplus \overleftarrow{h^t} \tag{27}$$

Here $\oplus$ is used for performing the element-wise addition operation, for forward direction hidden state $\overrightarrow{h^t}$, the time-series input from 1 to t is provided. While the hidden state for backward direction $\overleftarrow{h^t}$ is computed by feeding the reverse time-series input from t to 1.

4. Proposed MAG-D architecture for VM consolidation

The proposed MAG-D architecture is illustrated in Fig. 8. It takes input the resource utilization dataset of VMs containing spatial time steps data and forecasts the future CPU and memory usage values in different time steps. Initially, the methodology employs data preprocessing (moving average, scaling etc.) and transformation (Eqs. (1) and (2)) steps to generate the data in the required representation for input to the learning models. Subsequently, the proposed strategy involving clustering and prediction algorithms is applied to forecast the future resource usage demand. The intuition behind incorporating a clustering algorithm is to efficiently and accurately categorize similar usage patterns or workload intensity samples into groups. This will help the prediction models better delineate non-linear variations usage patterns present in the data, thus returning better prediction accuracy. An in-detailed explanation of the various phases present in the proposed architecture are discussed as follows:

4.1. Data pre-processing

To solve workload prediction problems requires training models with past resource usage data characterizing the PM (Physical Machine) and VM (Virtual Machine) resource usage behavior. Nevertheless, in most of the scenarios, not all of the time instances record the overall PM and VM utilization. Consequently, applying the feature engineering process to identify trivial jobs is suggested to minimize the model's memory and computational cost. For efficient cloud resource usage forecasting, we have used multivariate analysis. This technique applies all relevant resource usage metrics to identify the influence of all features on the target metric. With feature engineering, the time-series decomposition is performed, which helps to identify the data trends and the correlation between them. The time-series decomposition applies the moving average process to investigate the trends and seasonal behaviors. In addition, the estimation of error components helps to manage the undetermined events present in the time-series data of resource usage.

4.1.1. Data normalization

The Google cluster's traces comprises the workloads of eight compute clusters. It keeps the information of all job submissions, the decision of scheduling, and resource utilization data of each job running on an individual cluster [33]. The resource usage information is recorded for each 5-min period. We have dropped the rows comprising the missing and incomplete information. These resource usage data correspond to different scales. Therefore, if we feed the data directly to the forecasting model, it may result in out-of-proportion weights for different resource usage metrics. It can further slow down the overall learning of the deep learning network. Hence, we have performed the data normalization using the min–max scaling approach to transform the input resource usage data into some standard scale, i.e., range [0,1].

4.2. Proposed multivariate attention bidirectional GRU network

Algorithm 1 illustrates the step-by-step phases of the MAG-D prediction model. At-first it performs the data preprocessing (explained in Section 4.1), data transformation (Eqs. (1) and (2))

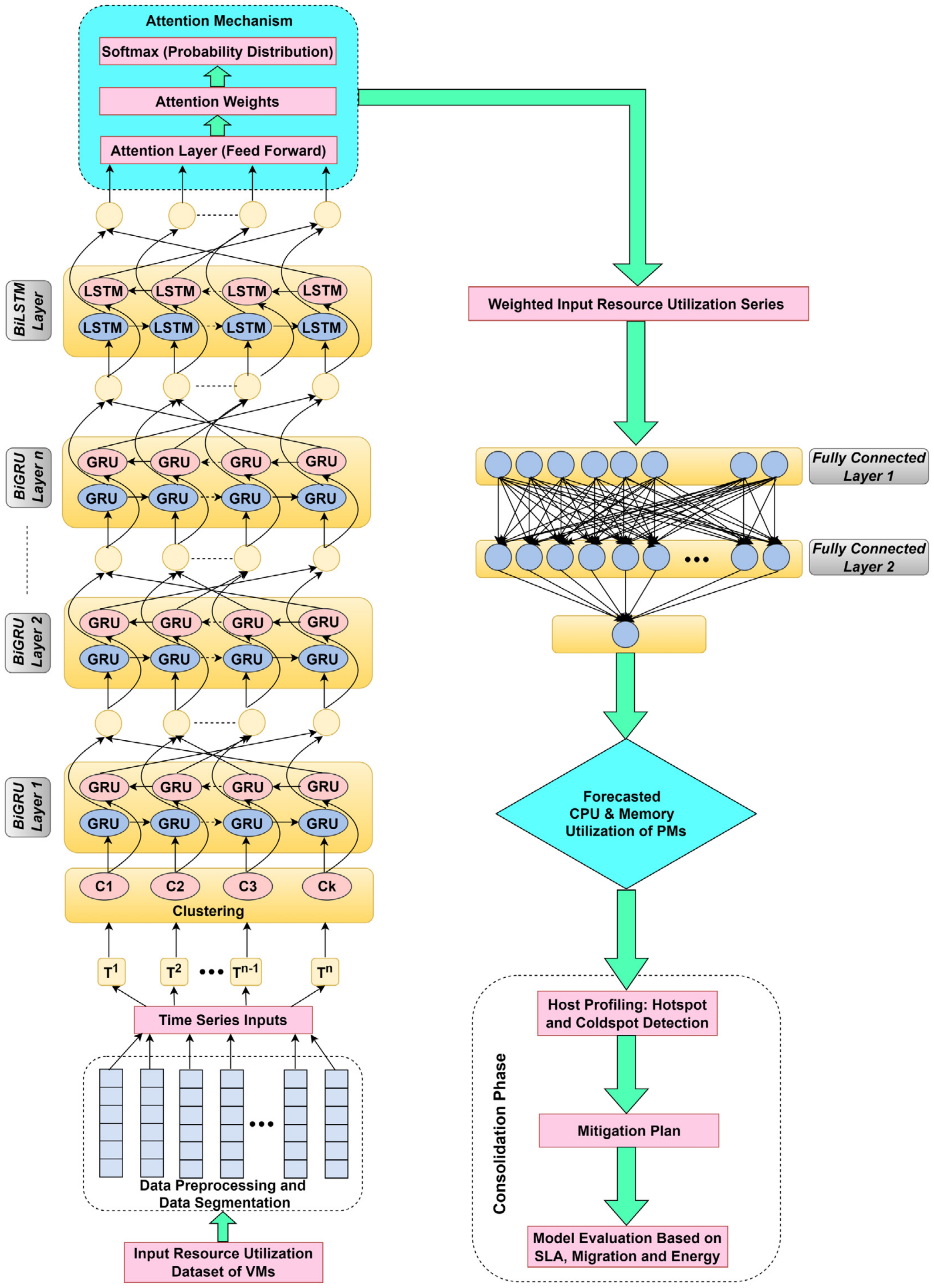

Attention Mechanism
Softmax (Probability Distribution)
Attention Weights
Attention Layer (Feed Forward)
BiLSTM Layer
LSTM
BiGRU Layer n
BiGRU Layer 2
BiGRU Layer 1
GRU
C1
C2
C3
Ck
Clustering
T1
T2
Tn-1
Tn
Time Series Inputs
Data Preprocessing and Data Segmentation
Input Resource Utilization Dataset of VMs
Weighted Input Resource Utilization Series
Fully Connected Layer 1
Fully Connected Layer 2
Forecasted CPU & Memory Utilization of PMs
Consolidation Phase
Host Profiling: Hotspot and Coldspot Detection
Mitigation Plan
Model Evaluation Based on SLA, Migration and Energy

Fig. 8. MAG-D architecture for VM consolidation.

and data segmentation [Algorithm 1: Steps 3 and 4 performed using Eqs. (3)–(6)] to form the time series inputs.

Further, the clustering analysis [Algorithm 1: Step 5] is implemented on the time series inputs, which recognizes the group with similar intra-cluster patterns and varying inter-clusters patterns. The current research study employs k-means clustering approach with DTW (Dynamic Time Warping) distance measure to group input time series of the train set into G number of clusters [Algorithm 1: Step 5]. The reason behind using DTW distance measure is to efficiently utilize the time-varying characteristics patterns of the input time series to identify clusters, which might not be feasible through several other available distance measures such as Euclidean, Manhattan etc. Another critical aspect of the k-means algorithm is to determine the optimal value of k (number of clusters). In the current study, the value of the critical clustering parameter k is determined using the elbow method. Fig. 9 demonstrates the results of the elbow method applied to the input resource usage dataset. From Fig. 9, it can be seen that the optimal 'k' value for the dataset used in the present study is 3. So, the k-means clustering with DTW is applied to

Algorithm 1: Proposed Algorithm

```
1  Input: Input Time-series $T = \{T_1, T_2, \ldots, T_n\}$ where
     $T_i = \{T_{i1}, T_{i2}, \ldots, T_{ix}\}$ here, $n$ represents the samples and
     $x$ represents the features. Total iterations $N$;
2  Output: Proposed MAG-D model
3  Training $Train_{T_i}$= $T_i[0 : tr, :]$ where tr= len $(T_i)$ * 0.80
4  Testing $Test_{T_i}$= $T_i[tr :, :]$
5  Apply Clustering process with DTW distance measure to
     cluster input time-steps $Train_{T_i}$ into $G$ clusters
     $\{C_1, C_2, \ldots, C_G\}$.
6  for each cluster $C_i$:
7  begin
8      Divide the time-series into two phases:
9      Training $Train_{C_i}$= $Train_{T_i}[0 : tr, :]$ where tr= len
         $(Train_{T_i})$ * 0.90
10     Testing $Test_{C_i}$= $Train_{T_i}[tr :, :]$
11     Train a prediction model amalgamating Bi-GRU and
         Bi-LSTM with Attention Mechanism
12     begin
13         $O_1 \leftarrow Bi - GRU(W_1, Train_{C_i})$
14         $O_2 \leftarrow Bi - GRU(W_2, O_1)$, where $W_2$ represents the
             weights at layer 2.
15         $O_3 \leftarrow Bi - GRU(W_3, O_2)$
16         $O_4 \leftarrow Bi - LSTM(W_4, O_3)$
17         $O_5 \leftarrow Attention(W_5, O_4)$
18         $O_6 \leftarrow Dense(W_6, O_5)$
19         $O_7 \leftarrow Dense(W_7)$, where $O_7$ is the target output
20         Calculating the difference of actual and forecasted
             values
21         Adjusting the weights of final layer and
             intermediate hidden layers using
             back-propagation strategy
22         Until the change in the error value during
             successive iterations falls below the threshold
             value (0.001)
23 For each $test_{sample} \in Test_{T_i}$
24 begin
25     Computes the similarity with each Cluster $C_i$ where i $\in$
         (0, G)
26     Identify the cluster $C_s$ with maximum similarity
27     Use the Cluster model for test sample prediction.
28 Evaluate the forecasting performance of MAG-D approach
     on complete test dataset.
```

identify three groups in the dataset. For out-of-sample prediction in each cluster, we have divided the time series into two parts [Algorithm 1: Steps 8–10]: (i) the training set having 90% of resource usage samples and (ii) the test set having 10% of resource usage samples. The training set comprising training (80%) and validation data (10%) is used for constructing the prediction models. For building the deep learning architecture, we have integrated the Bi-GRU and Bi-LSTM with the attention mechanism [Algorithm 1: Steps 13–21], which helps to extract relevant input time series segments at individual timestamps and perform the weight allocation process. It manages the temporal sequences present in the dataset through such complex hidden feature extraction. Each cluster is trained by applying the stacked Bi-GRU and BiLSTM deep network learners. These stacked deep learner, first comprises n Bi-GRU layers, which works as the primary feature-learning layer, followed by an additional Bi-LSTM layer is used for exhaustive and complex feature learning [Algorithm 1: Steps 13–17]. The stacked networks can perform multi-step-ahead forecasting using the past utilization of resource traces.

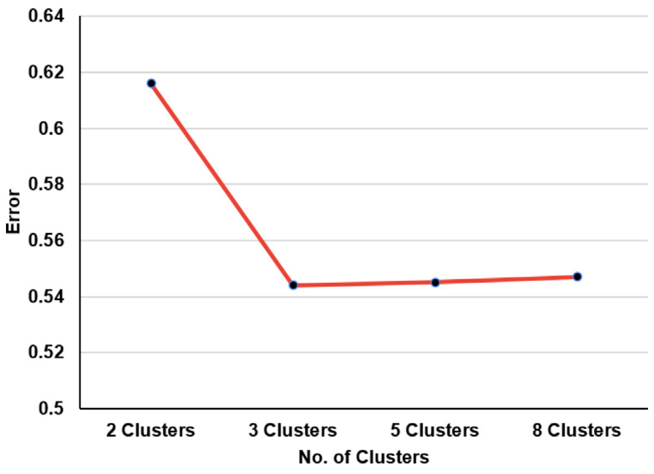


Fig. 9. Analysis for optimal value of k using Elbow method.

Moreover, the extracted weighted input resource utilization series are then fed to the fully connected network layers [Algorithm 1: Steps 18–19]. Once we get the target output, the model computes the difference between the actual and forecasted resource usage values to improve the model's performance. It adjusts the final layer's weights and intermediate hidden layers using the back-propagation technique [Algorithm 1: Step 21]. To evaluate the model's performance, we have the 10% of the test set comprising the multivariate resource usage values. For each test sample of the test set, it computes the similarity with each cluster to identify the cluster having maximum similarity [Algorithm 1: Steps 25–26] and then applies the cluster model for test sample prediction [Algorithm 1: Step 27]. The overall performance of the prediction model is evaluated on the complete test dataset (Algorithm 1: Step 28).

4.3. Analysis of computational complexity

For estimating the overall computational complexity of proposed model, we adopted the fundamental model of neural network with R cells of hidden layer, V and Y number of inputs, and outputs respectively [43,46]. Now, the total parameters for a simple multi-layer perceptron (MLP) comprising a single hidden layer expressed as: $TP_{MLP} = VR + RY$. In the case of RNN, the total parameters for all the hidden layer cells having RNN units, can be expressed as: $TP_{RNN} = VR + R^2 + RY$, here R^2 is used for recurrent connection. For LSTM cell having three gates with cell state, the total parameters expressed as $TP_{LSTM} = 4VR + 4R^2 + 3R + RY$. Now, for a GRU cell having architecture of 2 gates with a hidden state, the number of parameters are: $TP_{GRU} = 3VR + 3R^2 + 3R + RY$. For two stacked BiLSTM layers, the total parameters will be: $TP_{BiLSTM} = 2 * P_{LSTM}$. Similarly for M number of layers, the complexity is: $SP_{BiLSTM} = M * P_{LSTM}$. For bi-directional GRUs (BiGRU), we can represent the overall parameters as $TP_{BiGRU} = 2 * P_{GRU}$. For N number of stacked BiGRU layers, the total parameters are expressed through $SP_{BiGRU} = N * P_{GRU}$. In the similar way, the total parameters of attention layer can be expressed as $TP_{att} = V^2 * R$. Similarly for J stacked attention layers, the total parameters will be $SP_{att} = J * TP_{att}$. For K fully connected layers, we can express the total parameters of stacked fully connected layers as: $SP_{FC} = K * TP_{MLP}$. The architecture of MAG-D network comprises three stacked BiGRU layers, one stacked Bi-LSTM layer, one attention layer, followed by two fully connected layers. Therefore, the generalized parameters for proposed network is: $T_{MAG-D} \approx SP_{BiGRU} + SP_{BiLSTM} + SP_{att} + SP_{FC} + in * TP_{MLP}$. Here, we used in to express the total layers for intermediary features extraction and aggregation for forwarding in the K fully connected layers. For ϱ training epochs, the computational complexity of MAG-D architecture will be: $Comp_{MAG-D} \approx O(T_{MAG-D} * \varrho)$. If we assume the same input layer for all stacked networks. Finally, the computational complexity of the MAG-D architecture for ϱ training epochs will be: $Comp_{MAG-D} \approx O(R(R+Y) * \varrho)$.

5. Result discussions

5.1. Workload data

For performance validation, we have used the real traces of Google cluster [33]. The workload data of the Google cluster has 12,500 PMs recorded in May 2011. This trace provides a complete insight into the real cloud data center environment. In general, the workload arrives at the cluster as jobs. It has the running-time traces for more than 650 thousand real-time jobs performing different scheduling operations, including start, end and execution times for 29 days. These jobs are scheduled on heterogeneous physical machines with different cores and RAM. Each job in the cluster trace is associated with a set of resource usage metrics collected at different periods. The resource tables of all the traces are divided into three classes, namely, Jobs & Tasks, Machines, and Resource utilization. In which different features accompany each class. For model testing, we have used the resource usage table only. A brief summary of these resource usage metrics is tabulated in Table 1. The tabulated resource metrics are available in the resource usage table of Google cluster trace. The resource usage values are accumulated at each 10-second time-steps. The resource usage table has 20 columns and 12 types of resource usage features. CPU usage is one of the most crucial resource metrics. If jobs are taking longer to run than expected, then it is typically the first place to observe. Memory usage forecasting plays a vital role in the case of memory-intensive applications. This work primarily focused on predicting future CPU usage and memory utilization trends. Although, the proposed approach can be extended for forecasting other resource metrics as well. We have adopted univariate, bi-variate, and multivariate predictions with Google cluster traces. For multivariate CPU and memory usage prediction, all resource metrics are used. For univariate CPU usage prediction, only the CPU usage metric is adopted. In the case of univariate memory usage prediction, only the memory usage metric is considered. For bivariate CPU and memory usage prediction, we have considered both memory and CPU usage metrics. During the pre-processing data phase, we dropped the rows with incomplete and missing information. Moreover, we have consolidated the rows to transform the 5 min of usage data for a single row of forecasting. For deep learning models training, in total, 86,880 samples comprising ten days of total workload are utilized. The dataset is divided into training, validation, and test sets. During the validation phase, hyper-parameter tuning is performed. Afterwards, we have selected the best fitting parameters for the individual model for future workload forecasting in the next 90, 180, and 270 steps, respectively. These three prediction horizons are useful for analyzing out-of-sample predictions. Generally, the computational load of a cloud data center is highly fluctuating and time-varying in nature; thus, if a forecasting model captures trends for one forecasting horizon may not be suitable for another forecasting horizon [53]. Therefore, in this study, we have analyzed three prediction horizons to obtain the best one. A machine with Ubuntu 20.04, 64-bit, 48 cores Intel Xeon Gold Processor, 256 GB RAM and A4000 GPU is used for training of models.

Table 1
Resource utilization metrics.

Type of metrics	Description
CPU	Aggregated CPU utilization
MEM	Aggregated memory usage
MAXM	Maximum memory usage observed
VM	Allocated memory
MAXC	Maximum CPU usage
TPC	Total page cache utilization
CPI	CPI over all nodes
UPC	Aggregated unmapped page cache
MAI	Memory accesses per instruction
MAXD	Maximum disk I/O time detected
DSP	Disk capacity space usage
DIO	Disk I/O time over all disks

5.2. Performance parameters

Estimating the prediction capability and reliability of the prediction models is a critical task. In this research work, we have utilized two widely adopted performance metrics for MAG-D approach efficacy validation, namely MAE: Mean Absolute Error and RMSE: Root Mean Squared Error. The details of these listed measures are given as follows:

- RMSE: determines the quality of the predictions by measuring the standard deviation of the error. Mathematically, it is formulated as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(a_r^t - p_r^t)^2}{T}} \tag{28}$$

  where r represents the resource usage metric (CPU or Memory), T represents the prediction time horizon or number of samples to consider (90, 180 and 270 steps), and the actual values & predicted values at timestamp t are denoted as a^t and p^t respectively.
- MAE: estimates the quality of predictions by determining the average magnitude of forecasting errors. The mathematical equation for MAE is given as:

$$MAE = \frac{\sum_{t=1}^{T}|a_r^t - p_r^t|}{T} \tag{29}$$

The current research work implies the repetitive multi-step forecasting to predict the usages of CPU & memory resources at three different time horizons.

5.3. Experimental results and discussions

5.3.1. Effect of network parameters

Parameters that control the learning or define the architectural representation of a neural model are termed hyper-parameters. They are critical as their value may impact the overall generalization capability, learning and forecasting accuracy of the target model. Deciding the best/optimal value of hyper-parameters for a given problem is a challenging task. There are no universal hyper-parameters' value that can be well-adopted for solving research problems pertaining to different application domains or datasets. In this context, a process employed to determine the optimal value of models' parameters is termed hyper-parameter tuning. There are different ways to perform hyper-parameters tuning, such as random search, grid search, bayesian optimization etc. In the current study, the grid search strategy is employed to estimate the optimal hyper-parameters value of the developed neural models. The strategy begins by defining the possible values of all hyper-parameters related to a target model. Subsequently, the model is trained for all possible combinations

Table 2
Hyper-parameters estimation for MAG-D.

Activation function : ReLU						
RMSE/Batch size	CPU usage (In steps)			Memory usage (In steps)		
	90	180	270	90	180	270
16	**0.00506**	**0.00511**	**0.00514**	0.00382	0.00385	0.00385
32	0.00522	0.00527	0.00529	**0.00378**	**0.00380**	**0.00381**
64	0.00510	0.00516	0.00518	0.00388	0.00390	0.00391
128	0.00514	0.00519	0.00521	0.00380	0.00383	0.00384
256	0.00521	0.00526	0.00528	0.00389	0.00390	0.00392
Activation function : Tanh						
RMSE/Batch size	CPU usage (In steps)			Memory usage (In steps)		
	90	180	270	90	180	270
16	0.00514	0.00518	0.00520	0.00397	0.00399	0.00399
32	0.00513	0.00517	0.00519	0.00394	0.00396	0.00397
64	0.00510	0.00515	0.00517	0.00390	0.00392	0.00393
128	0.00515	0.00520	0.00522	0.00387	0.00389	0.00390
256	0.00511	0.00515	0.00517	0.00403	0.00404	0.00404

Table 3
Hyper-parameter specification.

Architecture Details	
Name of parameter	Range
Input layer neurons	Depends on input features dimension
BiLSTM layer neurons	[4–60]
Final layer (Dense) neurons	1
Training step/epochs	[70–400]
Dropout rate	[0.01–0.3]
Batch size	[16–256]
Optimizer	adam
Activation function	ReLU/Tanh
Loss function	Mean squared error

of hyper-parameters values, and the corresponding error values are evaluated. Finally, the combination of hyper-parameters value with the least error is selected for the model building and prediction activities. To demonstrate the working of the grid approach, an example representation considering two hyper-parameters (Activation Function and Batch Size) is presented. Initially, the possible values for both parameters are specified i.e., Activation Function: {'*Tanh*', '*ReLU*'} and Batch Size: {16, 32, 64, 128, 256}. Subsequently, the target Proposed model is trained for all possible combinations of both hyper-parameters values, and the corresponding RMSE error values are recorded. Table 2 list the RMSE value obtained for the different combinations of these hyper-parameters values. From the results listed in Table 2, the following observations are drawn:

- Performance comparison achieved by two activation functions combined with different batch sizes is highly comparable. However, the best performance has been achieved by employing the '*ReLU*' activation function in both CPU usage and Memory usage prediction tasks.
- Increasing Batch size ($>$64) has resulted in increased RMSE for both '*Tanh*' and '*ReLU*' activation functions. The best performance has been achieved by keeping $batch\_size = 16$ in case of CPU usage prediction. A similar trend in RMSE variations is observed in the memory usage prediction model, and the best results were obtained by setting *batch_size* equal to 32.

Finally, the hyper-parameters ('*ReLU*' + *batch_size*_16 and '*ReLU*' + *batch_size*_32) combinations with least RMSE are selected for the target model building and prediction tasks. The highlighted value in Table 2 represents the best models' (least RMSE values) hyper-parameters. The aforementioned grid search strategy has been adopted to determine the optimal value of all hyper-parameters of the proposed approach. The details of hyper-parameters related to the proposed MAG-D approach and corresponding values are listed in Table 3.

5.3.2. Comparison with other recurrent neural networks

This section entails discussing the performance of the various forecasting models at the CPU usage and Memory usage prediction tasks. In the current study, we have developed eight prediction models, namely ARIMA [13], Linear Regression [43], GRU [43], LSTM [20,38] Bi-LSTM [43], Bi-GRU [46], CNN integrated LSTM (CNN + LSTM) [40] and proposed Approach (MAG-D) for the target prediction tasks. Moreover, three variants of each model have been developed and trained covering different featural aspects of the input dataset, i.e. uni-variate, bi-variate and all features. Hence, a total of $(8 * 3) = 24$ models, including traditional and deep neural models, were built for conducting the comparative performance evaluation. The performance investigation of these models is done for out-of-the-sample CPU utilization & Memory usage prediction at three different time horizons (i.e., 90, 180, and 270 steps). The comparative analysis of the models is performed by utilizing two popular performance metrics (RMSE and MAE) discussed in the section. The corresponding evaluation results of the different models are listed in Tables 4–9. Tables 4 and 5 summarize the prediction results of the eight models considering all features of the CPU usage and Memory usage tasks. The prediction results are presented in terms of RMSE (Table 4) and MAE values (Table 5).

Similarly, Tables 6 and 7 represents the prediction result of the uni-variate prediction models on the target CPU usage and Memory usage tasks. The prediction performance is estimated and represented at three different time intervals, i.e., 90 steps, 180 steps and 270 steps ahead. Following this pattern, the results of bi-variate prediction models in terms of RMSE and MAE prediction errors are listed in Tables 8 and 9, respectively. From the comparative evaluation of RMSE and MAE results listed in Tables 4–9, it has been investigated that the proposed MAG-D approach outperforms all other conventional and deep neural models considering different sets of features (uni-variate, bi-variate and all features). However, the proposed approach (MAG-D) has achieved the best prediction performance while considering all features aspects of the Google cluster dataset. Also, we have observed that as the step size increases, the prediction performance of all developed models degrades. The possible reason behind this is the accumulation of errors from performing multi-steps ahead execution of out-of-the-sample forecasting.

Furthermore, to have an in-depth analysis of the model performance, we have plotted radar graphs representing the results

Table 4
RMSE analysis for all features.

RMSE	CPU usage (In steps)			Memory usage (In steps)		
Model	All features					
	90	180	270	90	180	270
ARIMA [13]	0.00762	0.00773	0.00777	0.00535	0.00535	0.00536
Linear Regression [43]	0.00531	0.00535	0.00537	0.00424	0.00424	0.00425
GRU [43]	0.00539	0.00545	0.00547	0.00409	0.00411	0.00412
LSTM [38]	0.00533	0.00537	0.00539	0.00411	0.00413	0.00414
BiLSTM [20]	0.00539	0.00544	0.00547	0.00411	0.00413	0.00414
BiGRU [46]	0.00527	0.00534	0.00537	0.00390	0.00393	0.00395
CNN + LSTM [40]	0.00529	0.00534	0.00536	0.00399	0.00403	0.00404
Proposed approach	**0.00506**	**0.00511**	**0.00514**	**0.00378**	**0.00380**	**0.00381**

Table 5
MAE analysis for all features.

MAE	CPU usage (In steps)			Memory usage (In steps)		
Model	All features					
	90	180	270	90	180	270
ARIMA [13]	0.00513	0.00515	0.00516	0.00313	0.00314	0.00315
Linear Regression [43]	0.00382	0.00383	0.00383	0.00262	0.00262	0.00263
GRU [43]	0.00383	0.00383	0.00383	0.00275	0.00275	0.00275
LSTM [38]	0.00390	0.00390	0.00391	0.00279	0.00280	0.00280
BiLSTM [20]	0.00388	0.00389	0.00390	0.00281	0.00282	0.00282
BiGRU [46]	0.00369	0.00372	0.00373	0.00248	0.00250	0.00251
CNN + LSTM [40]	0.00390	0.00391	0.00392	0.00256	0.00257	0.00258
Proposed approach	**0.00349**	**0.00351**	**0.00352**	**0.00228**	**0.00230**	**0.00230** -

Table 6
RMSE analysis for univariate.

RMSE	CPU usage (In steps)			Memory usage (In steps)		
Model	Univariate					
	90	180	270	90	180	270
ARIMA [13]	0.00667	0.00674	0.0064	0.00488	0.00488	0.00485
Linear Regression [43]	0.00553	0.00557	0.00559	0.00475	0.00475	0.00476
GRU [43]	0.00555	0.00559	0.0056	0.00449	0.0045	0.0045
LSTM [38]	0.00555	0.00559	0.00561	0.00451	0.00451	0.00452
BiLSTM [20]	0.00555	0.00559	0.00561	0.00450	0.0045	0.00451
BiGRU [46]	0.00555	0.00559	0.00560	0.00451	0.00452	0.00451
CNN + LSTM [40]	0.00586	0.00589	0.00590	0.00467	0.00466	0.00467
Proposed approach	**0.00553**	**0.00556**	**0.00558**	**0.00433**	**0.00434**	**0.00434**

Table 7
MAE analysis for univariate.

MAE	CPU usage (In steps)			Memory usage (In steps)		
Model	Univariate					
	90	180	270	90	180	270
ARIMA [13]	0.00461	0.00461	0.00463	0.00488	0.00489	0.00485
Linear Regression [43]	0.00394	0.00395	0.00395	0.00298	0.00298	0.00299
GRU [43]	0.00402	0.00402	0.00403	0.00292	0.00292	0.00293
LSTM [38]	0.00403	0.00404	0.00404	0.00287	0.00287	0.00288
BiLSTM [20]	0.00403	0.00404	0.00404	0.00287	0.00287	0.00288
BiGRU [46]	0.00397	0.00399	0.00399	0.00293	0.00294	0.00294
CNN + LSTM	0.00442	0.00443	0.00443	0.00320	0.00321	0.00321
Proposed approach	**0.00383**	**0.00385**	**0.00389**	**0.00264**	**0.00265**	**0.00265**

Table 8
RMSE analysis for bivariate.

RMSE	CPU usage (In steps)			Memory usage (In steps)		
Model	Bivariate					
	90	180	270	90	180	270
ARIMA [13]	0.01087	0.01113	0.01122	0.00545	0.00546	0.00546
Linear Regression [43]	0.00549	0.00553	0.00555	0.00473	0.00473	0.00473
GRU [43]	0.00546	0.0055	0.00552	0.00444	0.00445	0.00445
LSTM [38]	0.00542	0.00546	0.00548	0.00451	0.00451	0.00451
BiLSTM [20]	0.0054	0.00545	0.00546	0.00456	0.00456	0.00457
BiGRU [46]	0.00541	0.00545	0.00546	0.00445	0.00446	0.00447
CNN + LSTM [40]	0.00591	0.00594	0.00595	0.00464	0.00464	0.00465
Proposed approach	**0.00533**	**0.00538**	**0.00540**	**0.00429**	**0.00430**	**0.00430**

Table 9
MAE analysis for bivariate.

MAE	CPU usage (In steps)			Memory usage (In steps)		
Model	Bivariate					
	90	180	270	90	180	270
ARIMA [13]	0.00482	0.00484	0.00489	0.00314	0.00315	0.00316
Linear Regression [43]	0.00396	0.00395	0.00396	0.00296	0.00296	0.00296
GRU [43]	0.00395	0.00394	0.00394	0.00278	0.00279	0.00279
LSTM [38]	0.00396	0.00396	0.00397	0.00279	0.00279	0.00280
BiLSTM [20]	0.00395	0.00396	0.00396	0.00277	0.00277	0.00278
BiGRU [46]	0.00391	0.00392	0.00392	0.00274	0.00274	0.00275
CNN + LSTM [40]	0.00459	0.00459	0.00460	0.00309	0.00310	0.00310
Proposed approach	**0.00357**	**0.00358**	**0.00359**	**0.00246**	**0.00247**	**0.00247**

Table 10
Comparison of model training time (in ms).

Time (ms)	CPU			Memory		
Models	Univariate	Bivariate	Multivariate	Univariate	Bivariate	Multivariate
ARIMA [13]	1 180 000	1 230 000	1 590 000	1 149 000	1 273 000	1 446 000
Linear Regression [43]	27.78	24.44	52.499	16.25	26.90	79.90
GRU [43]	5	4	4	3	4	4
LSTM [38]	5	5	4	4	3	4
BiLSTM [20]	5	8	5	4	8	5
BiGRU [46]	7	8	8	7	5	8
CNN + LSTM [40]	4	4	11	5	4	8
Proposed approach	**3**	**4**	**3**	**2**	**3**	**3**

of all models developed in the present study. The charts are demonstrated in Figs. 10 and 11. Figs. 10 and 11 represent graphs depicting the RMSE and MAE errors of all models at three different timestamps ahead forecasting (90 steps, 180 steps and 270 steps), respectively. The triangular area covered by each model is representative of the model prediction accuracy. The more area covered by a model, the less accurate the model is at future patterns estimation. From Figs. 10 and 11, it can be easily observed that the triangular area covered by the proposed approach is significantly very less than all other models developed in the study. Hence, it can be concluded that the proposed MAG-D approach attains the best forecasting accuracy than all other existing models developed in the current work.

The present study also involves analyzing the aggregated performance of the MAG-D approach at the target prediction tasks. For this purpose, the stacked graphs representing the RMSE error of the different prediction models are presented in Fig. 12(a)–(c). These graphs demonstrate the aggregated performance of the models for out-of-the-sample forecasting at 90 steps, 180 steps and 270 steps ahead forecasting. These graphs clearly represent that the proposed approach has the least error, thus, validating the prediction performance and reliability of the MAG-D approach.

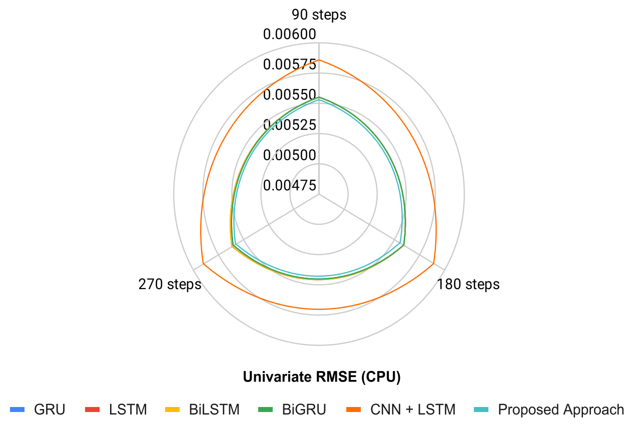
5.3.3. *Prediction results visualization on Google cluster dataset:*

The current section demonstrates the prediction plots of the best prediction model on the Google Cluster Trace dataset. From the comparative evaluation of the prediction models, it has been analyzed that the proposed MAG-D model has shown the best prediction accuracy for out-of-the sample prediction at different time horizons (90 steps, 180 steps and 270 steps). Hence, the prediction plots corresponding to the proposed MAG-D approach are presented in Fig. 13(a)–(f). These plots visualize the actual values and predicted output of the proposed approach on the CPU and memory usage tasks at different step sizes ahead forecasting. The blue line in these figures represents the actual output, and the orange line represents the generated output from the prediction model. The x-axis and y-axis of the figures represent the number of samples/timestamps and CPU/Memory usage, respectively. The figures show that the MAG-D approach effectively captures the sudden chaotic and non-linear variations of the trend patterns. Hence, the approach can be reliably employed in real-time scenarios to predict the CPU and Memory usage patterns at different step sizes.

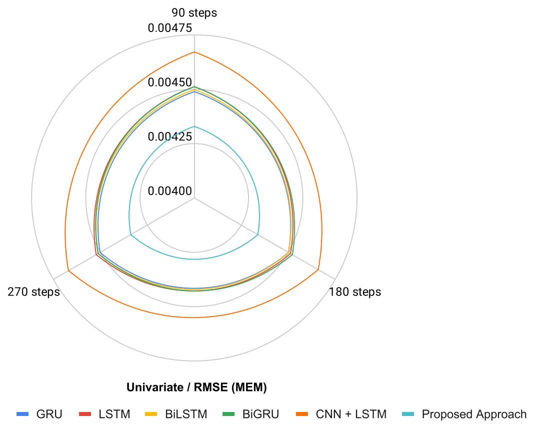
5.3.4. *Comparison of model training time*

In this subsection, we compare the training time of all the models. The experimental environment setup comprises Ubuntu 20.04, a 64-bit system with 48 cores Intel Xeon Gold Processor, 256 GB RAM and A4000 GPU. Table 10 presents the model training time of ARIMA, Linear Regression, GRU, LSTM, BiLSTM, BiGRU, CNN+LSTM, and the proposed approach. The training time is recorded for each epoch, and their mean value is presented in the table. We have compared the training time for univariate, bivariate, and multivariate prediction models for CPU and memory usage separately. As shown in Table 10, it is observed that the training time of ARIMA was longer than the other approaches. While, the Linear Regression takes more training time than the GRU, LSTM, BiLSTM, BiGRU, CNN+LSTM, and proposed approach. The model training time of the proposed approach is the least. In the case of CPU usage, it took about 3 ms, 4 ms, and 3 ms per epoch to train a model for univariate, bi-variate, and multivariate prediction models, respectively. In the case of memory usage, it took nearly 2 ms, 3 ms, and 3 ms per epoch to train a model for univariate, bi-variate, and multivariate prediction models correspondingly. In addition to the above-listed model training time, the proposed approach takes additional time (‘t seconds’) to perform single pass time-series clustering, which increases the total model building time of the proposed approach. However, as we perform clustering only once, the aforementioned executing requirements are relatively easy to accomplish, thus, making the proposed approach more applicable in practice.

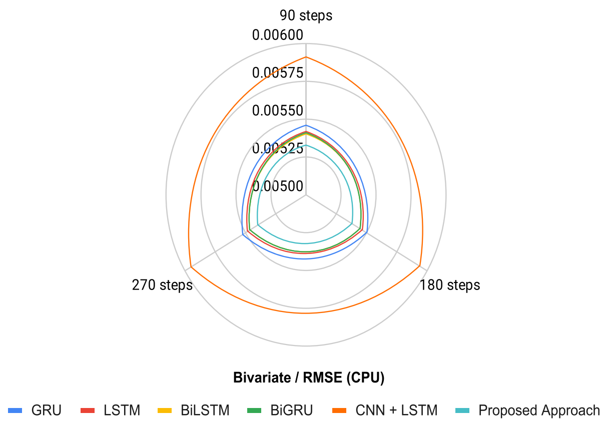
Discussion Regarding Model Retraining Requirements: Cloud workload forecasting is a complex and challenging task due to the non-linear and abrupt variations in the usage demand. The varying usage demand comprises input patterns from two major classes, namely known or estimated deviations (growth over the years) and unknown irregular demand patterns (completely



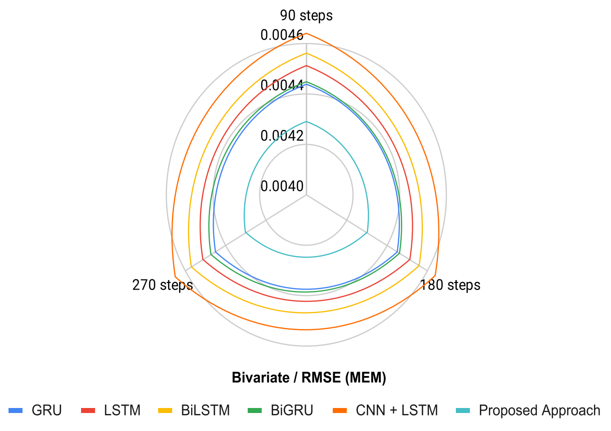
(a) CPU RMSE Univariate



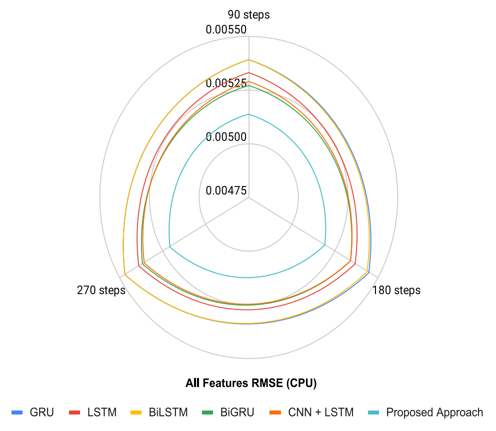
(b) Memory RMSE Univariate



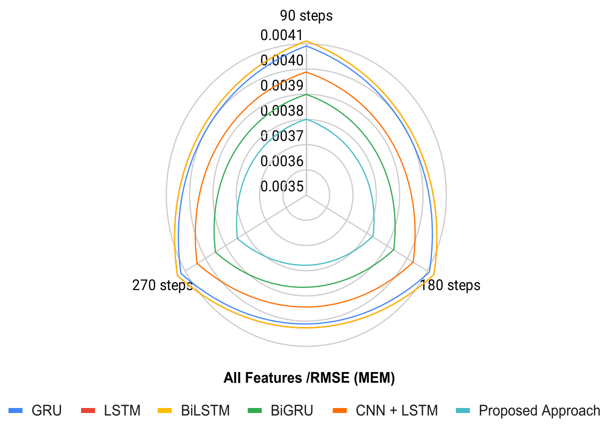
(c) CPU RMSE Bivariate Features



(d) Memory RMSE Bivariate Features



(e) CPU RMSE All Features



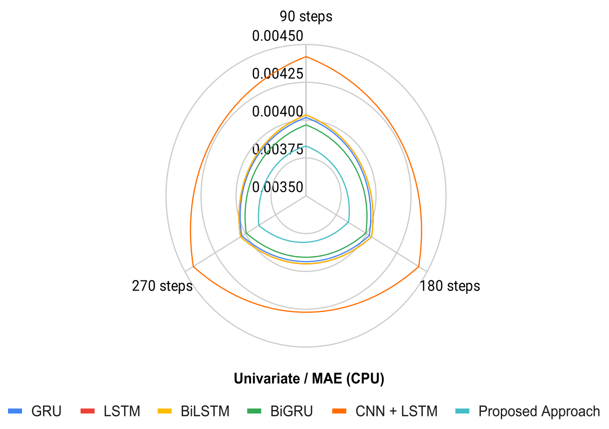
(f) Memory RMSE All Features

Fig. 10. Models' performances (RMSE) comparison.

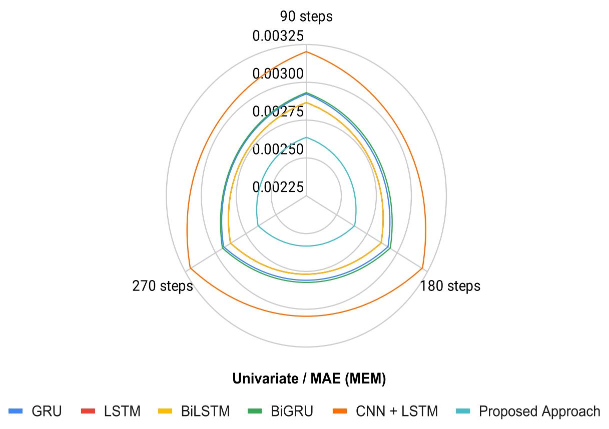
unpredictable such as the number of users increased during the COVID period). In the present approach, we proposed an efficient approach 'MAG-D' to estimate the future cloud workload patterns accurately. However, one critical aspect that needs attention in this context is model retraining, i.e. when the model needs to be retrained over the new incoming data to maintain desired reliability and accuracy. No global method exists to determine the exact time after which the retraining process must be executed. So, the machine learning or deep learning models need to be periodically updated with new data to maintain the desired performance. Besides this, there are several other factors on which the model retraining may depend, such as performance degradation over new data, changes in the target features or data (prediction variable) distribution patterns.

6. Conclusion

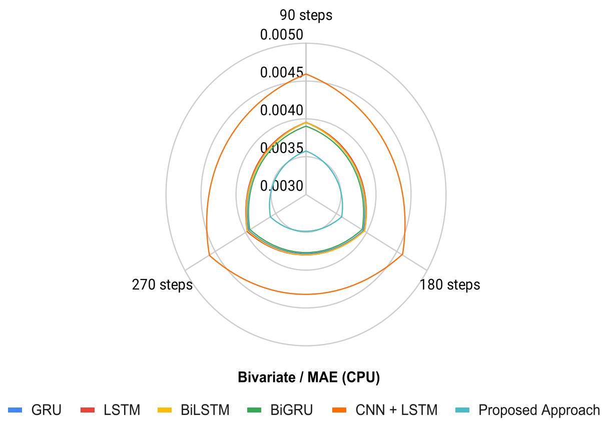
This study presents 'MAG-D', a multivariate attention bidirectional GRU-based deep learning architecture for multistep-ahead workload forecasting in cloud data centers. It takes the input



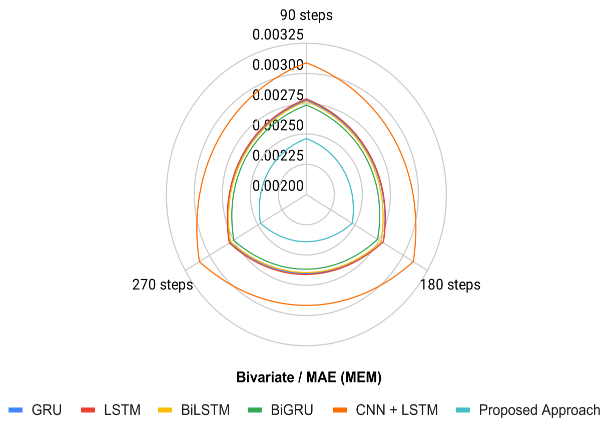
(a) CPU MAE Univariate



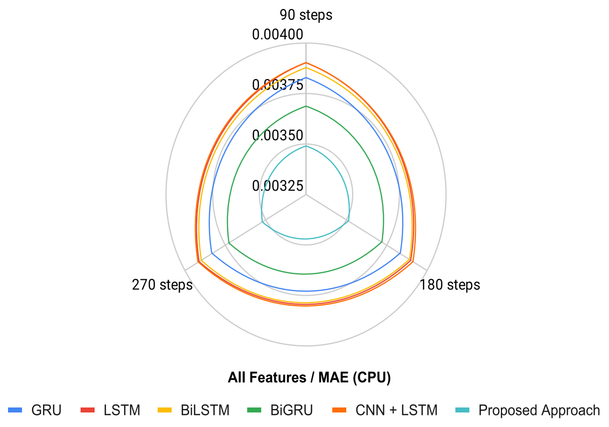
(b) Memory MAE Univariate



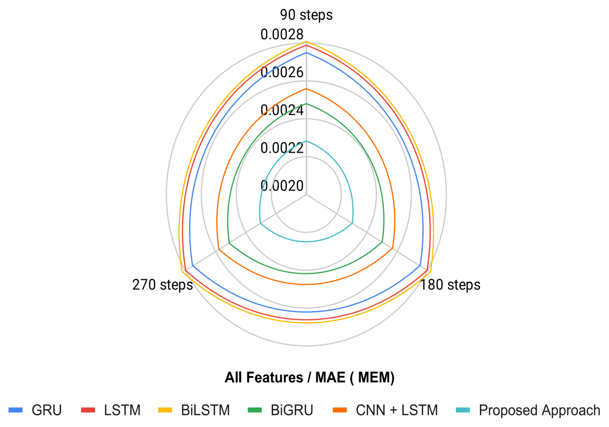
(c) CPU MAE Bivariate Features



(d) Memory MAE Bivariate Features



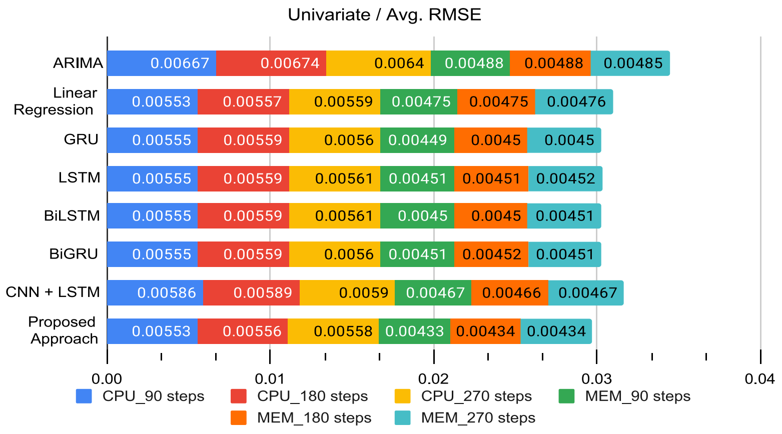
(e) CPU MAE All Features



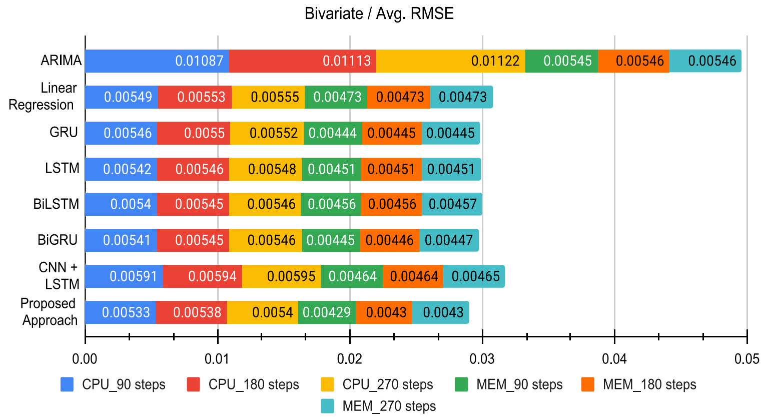
(f) Memory MAE All Features

Fig. 11. Models' performances (MAE) comparison.

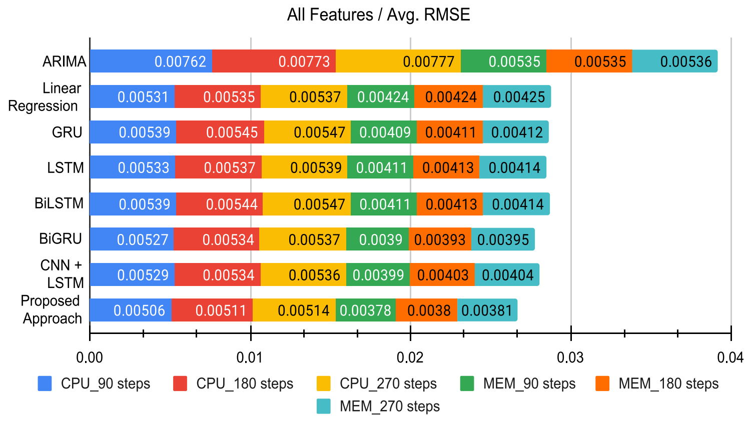
resource utilization dataset of VMs containing data of spatial time steps and forecasts the CPU and memory usage values in different time steps. For building the deep learning architecture, we have integrated the stacked Bi-GRU and Bi-LSTM with the attention mechanism, which helps to extract relevant input time series segments at individual timestamps and perform the weight allocation process. The stacked networks are capable of performing the multi-step-ahead cloud resource predictions implying the past resource utilization traces. Firstly, it forecasts the CPU and memory usage using the MAG-D deep learning architecture. The forecasting of future resource usage helps in VM consolidation during energy-efficient cloud resource management. This study performs trace-driven simulations utilizing Google's cluster



(a) Combined RMSE Univariate (CPU and Memory)



(b) Combined RMSE Bivariate (CPU and Memory)



(c) Combined RMSE All Features (CPU and Memory)

Fig. 12. Aggregated RMSE performance comparison for proposed approach with state-of-the-art prediction models'.

traces. The results are promising and achieve notable improvements over recent prediction models in terms of RMSE and MAE performance matrices. In future, we will extend this work to carbon emission reduction in geo-distributed cloud data centers.

CRediT authorship contribution statement

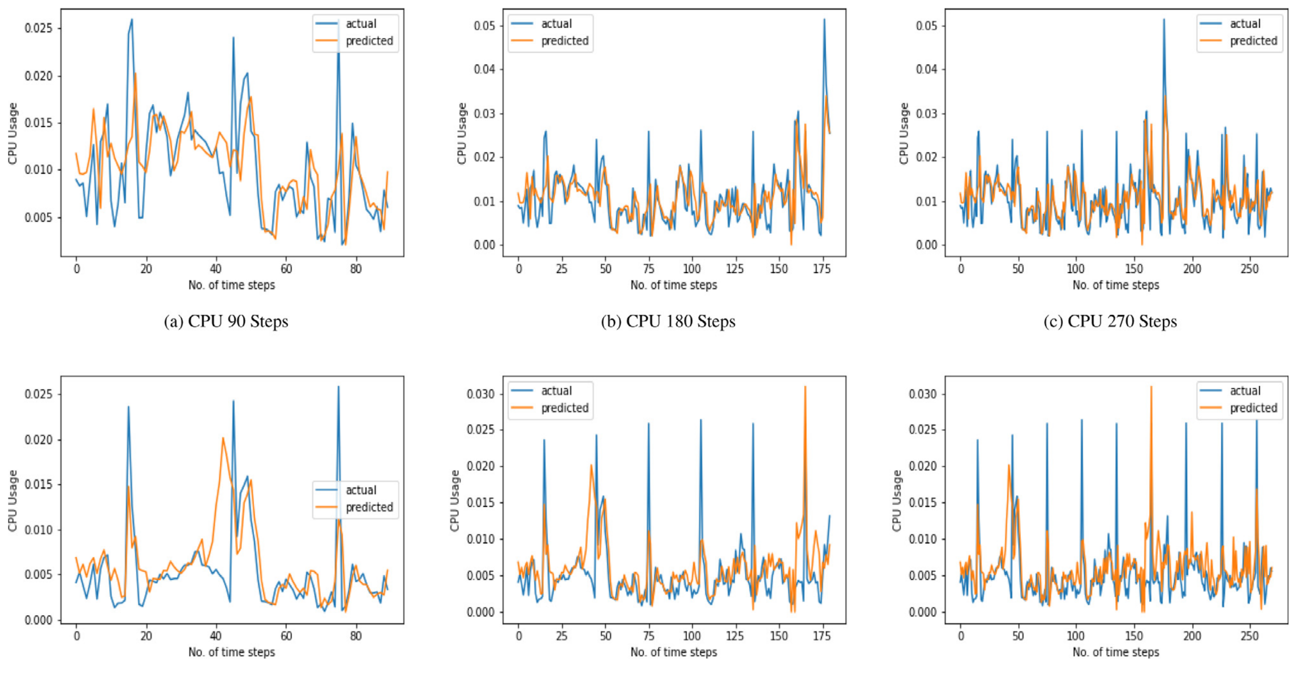
Yashwant Singh Patel: Conceived the study, Performed the numerical experiments, Analyzed the data, Wrote the manuscript. **Jatin Bedi:** Conceived the study, Performed the numerical experiments, Analyzed the data, Wrote the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data is publicly available.



(a) CPU 90 Steps (b) CPU 180 Steps (c) CPU 270 Steps

(d) Memory 90 Steps (e) Memory 180 Steps (f) Memory 270 Steps

Fig. 13. Visualization of prediction results of the proposed approach.

References

[1] J. Kumar, R. Goomer, A.K. Singh, Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters, Procedia Comput. Sci. 125 (2018) 676–682, http://dx.doi.org/10.1016/j.procs.2017.12.087.

[2] Q. Zhang, L.T. Yang, Z. Yan, Z. Chen, P. Li, An efficient deep learning model to predict cloud workload for industry informatics, IEEE Trans. Ind. Inform. 14 (2018) 3170–3178, http://dx.doi.org/10.1109/TII.2018.2808910.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality fordelivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.

[4] S. Jeddi, S. Sharifian, A hybrid wavelet decomposer and GMDH-ELM ensemble model for network function virtualization workload forecasting in cloud computing, Appl. Soft Comput. (2020) http://dx.doi.org/10.1016/j.asoc.2019.105940.

[5] A. Mozo, B. Ordozgoiti, S. Gómez-Canaval, Forecasting short-term data center network traffic load with convolutional neural networks, PLoS ONE (2018) http://dx.doi.org/10.1371/journal.pone.0191939.

[6] M. Amiri, L. Mohammad-Khanli, R. Mirandola, An online learning model based on episode mining for workload prediction in cloud, Future Gener. Comput. Syst. 87 (2018) 83–101, http://dx.doi.org/10.1016/j.future.2018.04.044.

[7] S. Sharifian, M. Barati, An ensemble multiscale wavelet-GARCH hybrid SVR algorithm for mobile cloud computing workload prediction, Int. J. Mach. Learn Cybern. 10 (2019) 3285–3300, http://dx.doi.org/10.1007/s13042-019-01017-.

[8] J. Kumar, A.K. Singh, R. Buyya, Ensemble learning based predictive framework for virtual machine resource request prediction, Neurocomputing (2020) http://dx.doi.org/10.1016/j.neucom.2020.02.014.

[9] Deborah Magalhães, et al., Workload modeling for resource usage analysis and simulation in cloud computing, Comput. Electr. Eng. 47 (2015) 69–81.

[10] Jatin Bedi, Durga Toshniwal, Energy load time-series forecast using decomposition and autoencoder integrated memory network, Appl. Soft Comput. 93 (2020).

[11] J. Bedi, Attention based mechanism for load time series forecasting: AN-LSTM, in: International Conference on Artificial Neural Networks, Springer, Cham, 2020, pp. 838–849.

[12] S. Subramanian, A. Kannammal, Real time non-linear cloud workload forecasting using the holt-winter model, in: Proc. 10th International Conference on Computing, Communication and Networking Technologies, ICCCNT, Kanpur, India, 2019, pp. 1–6.

[13] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, Joseph L. Hellerstein, Dynamic energy-aware capacity provisioning for cloud computing environments, in: Proc. International Conference on Autonomic Computing, ICAC, ACM, New York, NY, USA, 2012, pp. 145–154.

[14] V. Podolskiy, A. Jindal, M. Gerndt, Y. Oleynik, Forecasting models for self-adaptive cloud applications: A comparative study, in: Proc. IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems, SASO, Trento, Italy, 2018, pp. 40–49.

[15] Z. Gong, X. Gu, J. Wilkes, PRESS: PRedictive Elastic ReSource Scaling for cloud systems, in: Proc. International Conference on Network and Service Management, CNSM, IEEE, Niagara Falls, ON, Canada, 2010, pp. 9–16.

[16] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, J. Wilkes, AGILE: elastic distributed resource scaling for infrastructure-as-a-service, in: Proc. 10th International Conference on Autonomic Computing, ICAC, USENIX, San Jose, CA, 2013, pp. 69–82.

[17] Y.S. Patel, R. Misra, Performance comparison of deep VM workload prediction approaches for cloud, in: P. Pattnaik, S. Rautaray, H. Das, J. Nayak (Eds.), Progress in Computing, Analytics and Networking, in: Advances in Intelligent Systems and Computing, vol. 710, Springer, Singapore, 2018.

[18] S. Hochreiter, J.U. Schmidhuber, Long shortterm memory, Neural Comput. 9 (1997) 1735–1780.

[19] M. Ghorbani, Y. Wang, Y. Xue, M. Pedram, P. Bogdan, Prediction and control of bursty cloud workloads: a fractal framework, in: Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014, pp. 1–9.

[20] S. Gupta, A.D. Dileep, T.A. Gonsalves, A joint feature selection framework for multivariate resource usage prediction in cloud servers using stability and prediction performance, J. Supercomput. 74 (11) (2018) 6033–6068.

[21] Maryam Amiri, Leyli Mohammad-Khanli, Raffaela Mirandola, A sequential pattern mining model for application workload prediction in cloud environment, J. Netw. Comput. Appl. 105 (2018) 21–62.

[22] Li Ruan, et al., Workload time series prediction in storage systems: a deep learning based approach, Cluster Comput. (2021) 1–11.

[23] Jing Bi, et al., Integrated deep learning method for workload and resource prediction in cloud systems, Neurocomputing 424 (2021) 35–48.

[24] P. Singh, P. Gupta, K. Jyoti, TASM: technocrat ARIMA and SVR model for workload prediction of web applications in cloud, Clust. Comput. 22 (2) (2018) 619–633.

[25] Mohammad S. Aslanpour, et al., AutoScaleSim: A simulation toolkit for auto-scaling Web applications in clouds, Simul. Model. Pract. Theory 108 (2021) 102245.

[26] F. Caglar, A. Gokhale, iOverbook: Intelligent resource overbooking to support soft real-time applications in the cloud, in: Proc. IEEE 7th International Conference on Cloud Computing, CLOUD, IEEE, Anchorage, AK, USA, 2014, pp. 538–545.

[27] Sun-Yuan Hsieh, Cheng-Sheng Liu, Rajkumar Buyya, Albert Y. Zomaya, Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers, J. Parallel Distrib. Comput. 139 (C) (2020) 99–109, (May 2020).

[28] H. Wang, G. Li, G. Wang, J. Peng, H. Jiang, Y. Liu, Deep learning based ensemble approach for probabilistic wind power forecasting, Appl. Energy 188 (2017) 56–70.
[29] H. Assem, S. Ghariba, G. Makrai, P. Johnston, L. Gill, F. Pilla, Urban water flow and water level prediction based on deep learning, in: Computer Science (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNAI, vol. 10536, 2017, pp. 317–329.
[30] A.J. Hussain, D. Al-Jumeily, H. Al-Askar, N. Radi, Regularized dynamic self-organized neural network inspired by the immune algorithm for financial time series prediction, Neurocomputing 188 (2016) 23–30.
[31] S. Jeddi, S. Sharifian, A water cycle optimized wavelet neural network algorithm for demand prediction in cloud computing, Clust. Comput. 22 (2019) 1397–1412.
[32] L. Mo, L. Xie, X. Jiang, G. Teng, L. Xu, J. Xiao, GMDH-based hybrid model for container throughput forecasting: Selective combination forecasting in nonlinear subseries, Appl. Soft Comput. J. 62 (2018) 478–490, http://dx.doi.org/10.1016/j.asoc.2017.10.033.
[33] C. Reiss, J. Wilkes, J.L. Hellerstein, Google cluster-usage traces: format + schema, 2011, https://github.com/google/cluster-data.
[34] B. Song, Y. Yu, Y. Zhou, et al., Host load prediction with long short-term memory in cloud computing, J. Supercomput. 74 (2018) 6554–6568.
[35] K. Chakraborty, K. Mehrotra, C.K. Mohan, S. Ranka, Forecasting the behavior of multivariate time series using neural networks, Neural Netw. 5 (6) (1992) 961–970.
[36] P. Aboagye-Sarfo, et al., A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in Western Australia, J. Biomed. Inform. 57 (2015) 62–73.
[37] Y. Zhao, L. Ye, Z. Li, X. Song, Y. Lang, J. Su, A novel bidirectional mechanism based on time series model for wind power forecasting, Appl. Energy 177 (2016) 793–803.
[38] S. Gupta, D.A. Dinesh, Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks, in: Proc. International Conference on Advanced Networks and Telecommunications Systems, ANTS, IEEE, Bhubaneswar, 2017, pp. 1–6.
[39] G. Lai, W.C. Chang, Y. Yang, H. Liu, Modeling long- and short-term temporal patterns with deep neural networks, in: 41st International ACM SIGIR Conference Research Development Information Retrieval, SIGIR 2018, 2018, pp. 95–104.
[40] R. Zhao, R. Yan, J. Wang, K. Mao, Learning to monitor machine health with convolutional Bi-directional LSTM networks, Sensors (Switzerland) (2017).
[41] T. Zhongda, L. Shujiang, W. Yanhong, S. Yi, A prediction method based on wavelet transform and multiple models fusion for chaotic time series, Chaos Solitons Fractals 98 (2017) 158–172.
[42] J. Azar, A. Makhoul, R. Couturier, J. Demerjian, Robust IoT time series classification with data compression and deep learning, Neurocomputing (2020).
[43] Y.S. Patel, R. Jaiswal, R. Misra, Deep learning-based multivariate resource utilization prediction for hotspots and coldspots mitigation in green cloud data centers, J. Supercomput. (2021).
[44] Y.S. Patel, R. Jaiswal, S. Pandey, R. Misra, K Stacked bidirectional LSTM for resource usage prediction in cloud data centers, in: R. Misra, N. Kesswani, M. Rajarajan, V. Bharadwaj, A. Patel (Eds.), Internet of Things and Connected Technologies. ICIoTCT 2020, in: Advances in Intelligent Systems and Computing, vol. 1382, Springer, Cham, 2021.
[45] P. Yazdanian, S. Sharifian, E2LG: a multiscale ensemble of LSTM/GAN deep learning architecture for multistep-ahead cloud workload prediction, J. Supercomput. 77 (2021) 11052–11082.
[46] S. Behera, R. Misra, A. Sillitti, Multiscale deep bidirectional gated recurrent neural networks based prognostic method for complex non-linear degradation systems, Inform. Sci. 554 (2021) 120–144.
[47] P.J. Braspenning, F. Thuijsman, A.J.M.M. Weijters, Artificial Neural Networks: An Introduction to ANN Theory and Practice, Vol. 931, Springer Science Business Media, 1995.
[48] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (2) (1994) 240–254.
[49] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[50] A. Graves, S. Fernández, J. Schmidhuber, Bidirectional LSTM networks for improved phoneme classification and recognition, in: International Conference on Artificial Neural Networks, Springer, Berlin, Heidelberg, 2005, pp. 799–804.
[51] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014, arXiv preprint arXiv:1412.3555.
[52] Y. Su, C.C.J. Kuo, On extended long short-term memory and dependent bidirectional recurrent neural network, Neurocomputing 356 (2019) 151–161.
[53] Shaifu Gupta, Aroor Dinesh Dileep, Timothy A. Gonsalves, Online sparse blstm models for resource usage prediction in cloud datacentres, IEEE Trans. Netw. Serv. Manag. 17.4 (2020) 2335–2349.

Dr. Yashwant Singh Patel is currently working as an Assistant Professor in the Computer Science and Engineering Department (CSED) at Thapar Institute of Engineering & Technology, Punjab, India. He received his Ph.D. in Computer Science and Engineering from IIT Patna India, under the supervision of Prof. Rajiv Misra, in 2021. His research interests include resource allocation and consolidation problems for geo-distributed cloud datacenters. His current work is related to Network Function Virtualization, Cloud Computing, Edge Computing, IoT, Augmented Reality and Virtual Reality (AR/VR), and Deep Learning. Till now, he has published more than ten papers in reputed journals, two book chapters, one text book, and 22 papers in the proceedings of international conferences and workshops. He has also worked in a DST sponsored project undertaken by IIT Patna in course of PhD. He was also a recipient of Visvesvaraya Research fellowship.

Dr. Jatin Bedi is presently working as Assistant Professor in the Department of Computer Science and Engineering, Thapar University. He obtained his B.Tech. with distinction from Kurukshetra University, Kurukshetra, India, and M.Tech. degree from DCSA, Kurukshetra University, Kurukshetra, India. He received his Ph.D. degree from Indian Institute of Technology Roorkee, Uttarakhand, India in 2020. He has many publications in high impact SCI indexed journals and internationally reputed conferences including SIAM SDM, ICANN, ECML/PKDD, MLDM and many more.