Applied Intelligence

Efficient Kalman filter based Deep learning approaches for Workload Prediction in Cloud and Edge Environments

--Manuscript Draft--

Manuscript Number:	APIN-D-24-03382
Full Title:	Efficient Kalman filter based Deep learning approaches for Workload Prediction in Cloud and Edge Environments
Article Type:	Original Submission
Keywords:	Kalman Filter, Deep Learning, LSTM, Bi-LSTM, GRU, Attention, CNN, Edge Data Centers, Autonomous Vehicles
Corresponding Author:	Naveen Kumar M R National Institute of Technology Karnataka Mangalore, Karnataka INDIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	National Institute of Technology Karnataka
Corresponding Author's Secondary Institution:	
First Author:	Naveen Kumar M R
First Author Secondary Information:	
Order of Authors:	Naveen Kumar M R
	Annappa B, Ph.D.
	Vishwas Yadav
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	Offering cloud resources to consumers presents several difficulties for cloud service providers. When utilizing resources efficiently in cloud and edge contexts, precisely forecasting workload is a crucial problem. Accurate workload prediction allows intelligent resource allocation, preventing needless waste of computational and storage resources while meeting user's Quality of Service(QoS). In order to mitigate this issue, a hybrid novel Kalman filter with CNN and attention based Long Short Term memory (LSTM), Bi- directional Long Short Term Memory (BI-LSTM), and Gated Recurrent Unit (GRU) models are proposed for accurate prediction of workloads at Edge Servers. The proposed models were extensively evaluated on real world traces like Alibaba v2018, Materna, Bitbrains, Microsoft Azure 2019 and Planet lab datasets at various time intervals with and without using Kalman filter. The experimental comparison shows that 97%, 82% and 90% reduction in MSE for Alibaba, 73%, 73% and 63% reduction in MSE for Materna, 72%, 63% and 40% reduction in MSE for Planet lab, 95%, 77% and 96% reduction in MSE for Microsoft Azure and 91%, 87% and 91% reduction in MSE for Bitbrains using Kalman Filter and Convolutional LSTM with Attention (KF-CLA), Kalman Filter and Convolutional Bi-LSTM with Attention (KF-CBiLA) and Kalman Filter and Convolutional GRU with Attention (KF-CGA) respectively.

Efficient Kalman filter based Deep learning approaches for Workload Prediction in Cloud and Edge Environments

Naveen Kumar M R, Annappa B[†] and Vishwas Yadav[†]

Department of Computer Science and Engineering, National Institute of Technology, Surathkal, Mangalore, 575025, Karnataka, India.

*Corresponding author(s). E-mail(s): naveenkumarmr.227cs003@nitk.edu.in;
Contributing authors: annappa@ieee.org; vishwas.222cs034@nitk.edu.in;
[†]These authors contributed equally to this work.

Abstract

Offering cloud resources to consumers presents several difficulties for cloud service providers. When utilizing resources efficiently in cloud and edge contexts, precisely forecasting workload is a crucial problem. Accurate workload prediction allows intelligent resource allocation, preventing needless waste of computational and storage resources while meeting user's Quality of Service(QoS). In order to mitigate this issue, a hybrid novel Kalman filter with CNN and attention based Long Short Term memory (LSTM), Bi- directional Long Short Term Memory (BI-LSTM), and Gated Recurrent Unit (GRU) models are proposed for accurate prediction of workloads at Edge Servers. The proposed models were extensively evaluated on real world traces like Alibaba_v2018, Materna, Bitbrains, Microsoft Azure_2019 and Planet lab datasets at various time intervals with and without using Kalman filter. The experimental comparison shows that 97%, 82% and 90% reduction in MSE for Alibaba, 73%, 73% and 63% reduction in MSE for Materna, 72%, 63% and 40% reduction in MSE for Planet lab, 95%, 77% and 96% reduction in MSE for Microsoft Azure and 91%, 87% and 91% reduction in MSE for Bitbrains using Kalman Filter and Convolutional LSTM with Attention (KF-CLA), Kalman Filter and Convolutional Bi-LSTM with Attention (KF-CBiLA) and Kalman Filter and Convolutional GRU with Attention (KF-CGA) respectively.

Keywords: Kalman Filter,Deep Learning, LSTM, Bi-LSTM, GRU, Attention, CNN, Edge Data Centers, Autonomous Vehicles

1 Introduction

Cloud computing is being more widely developed and used by many businesses. A shared and dynamic pool of computing, storage, and network resources is created by integrating servers, storage services, software, and networks of Edge Data centers(EDCs) [1]. In addition, they leverage cloud and edge systems to dynamically assign memory, storage, and network bandwidth resources based on user demands. Cloud computing, supported by essential features including elasticity, on-demand services, accessibility, and a pay-as-you-use model, transforms service offerings by enabling resource and application sharing across users. Cloud systems frequently rely on the on-demand services offered by data centers[2]. On the other hand, poor resource utilization, increased power consumption, and unpredictable quality of service (QoS) are problems caused by workload patterns' intrinsic non-linearity, which calls for effective resource management and optimization techniques. In particular, idle servers waste electricity and resources, while overloaded servers deteriorate system performance. Thus, in order to manage resources efficiently, service providers need practical approaches. Predicting the fluctuating future workload and resource usage is major concerns of edge service providers and one such challenge is accurate workload and resource prediction at EDCs[3]. Estimating future workload needs accurate prediction, an essential method for well-informed decision-making, allowing for in-situ resource supply. Task suspension and longer service duration's may occur when numerous data processing tasks flood the edge data center at low CPU frequencies. On the other hand, extended idle times during low activity and lower CPU frequencies waste resources. However, the peril of resource overloading or under-allocation is reduced with efficient workload prediction techniques, minimizing needless overheads and improving resource utilization.

Many works in the previous have provided solutions for time series data predictions using statistical and Machine learning techniques like Auto Regressive Integrated Moving Average(ARIMA)[4] and Support Vector Machines(SVM)[5]. Recent advances in deep learning have given rise to models such as Long Short-Term Memory (LSTM) neural networks and Gated Recurrent Unit (GRU), which offer new ways to address the gradient disappearance issue with conventional Recurrent Neural Networks (RNNs) and achieve high-accuracy in time series prediction. Deep learning models have outperformed conventional state-of-the-art methods because of hierarchical features and learning representations. Convolutional Neural Networks (CNNs), stacked auto-encoders, and Deep Belief Networks (DBNs) are some of the deep learning techniques available. Many works have optimized parameters for multi-level feature extraction in deep learning systems using supervised and unsupervised methods. Investigations on Artificial Neural Networks (ANNs) gave rise to deep learning, a crucial component of Machine Learning (ML) and the foundation of the current generation of Neural Networks (NN). workload prediction within cloud environments, focus primarily on forecasting CPU utilization and memory utilization. The significance of accurately predicting these metrics cannot be overstated, as they serve as vital indicators of system performance and resource requirements. By leveraging historical data and employing advanced prediction models, this study aims to provide insights into workload patterns, enabling cloud and edge computing providers to make informed decisions regarding resource allocation and management strategies. Due to the fluctuating workload and resource utilization characteristics and the continued growth of data produced by EDCs throughout operations, traditional prediction methods cannot anticipate large-scale data with sufficient precision, leading to substandard outcomes. Henceforth, a novel Kalman filter with CNN and attention based LSTM, BI-LSTM, and GRU models been proposed for predicting resource utilization and achieving considerably better performance than baselines. To the best of our knowledge, no work has used the Kalman filter with CNN in deep learning models for workload prediction.Using a range of criteria, we have assessed five distinct cloud traces. Different features are present in every cloud trail. Our trials used conventional characteristics from Alibaba and Materna databases, including CPU and memory consumption. We only considered the CPU usage features for the Bitbrains, Microsoft Azure, and Planet lab datasets. We have incorporated cores and CPU with time as multivariate characteristics from Bitbrains, Materna and Microsoft Azure dataset. The following are the distinctions and contributions of the proposed method as compared to the previous work:

- In order to eliminate extreme points and noise interference from time series data, this research analyzes several smoothing techniques. After analyzing the results, it concludes that the Kalman filter is the best choice for attaining high prediction accuracy.
- Propose a novel Kalman filter with CNN and attention based LSTM, BI-LSTM, and GRU models to predict multivariate features future workload on edge servers. these models effectively extracts complex characteristics of time series and achieves high accuracy in predictions.
- A comparison of the suggested models for Alibaba and Materna traces with and without Kalman filter, having CPU and memory being assessed at various intervals of time such as 1, 5, 10, 30 and 60 minutes. Only CPU consumption for the Planet lab and Azure was considered into account and assessed at various intervals, including 10, 15, 30, and 60 minutes, respectively. We assessed the Bitbrains dataset at 1, 5, 15, 30, and 60 seconds since samples were captured every three seconds.
- All datasets are evaluated using ARIMA with and without Kalman filter, showing better performance than previous ARIMA-based methods.

2 Related works

2.1 Workload prediction using Conventional methods

A cloud workload prediction module that uses the ARIMA model to assign resources to Software as a Service(SaaS) providers has been proposed by authors. The model is evaluated using real-time traces; it attains an accuracy rate of up to 91%. This high degree of accuracy makes proactive resource allocation possible, guaranteeing effective use while protecting end users' QoS[4]. A hybrid ARIMA–ANN model is proposed to predict memory and CPU usage during cloud resource provisioning. The ARIMA model looks for linear cloud trace CPU and memory usage trends. On the other hand, the ANN improves prediction by boosting nonlinear components included in the residuals of the traces. The model forecasts consumption patterns and determines predicted values by mixing linear and nonlinear aspects. The study presents the over estimation rate and under estimation rate to reduce forecasting mistakes. Evaluation with BitBrain data and Google's 29-day trial shows how well the model predicts resource use, providing a thorough method for resource management in cloud environments[6]. A prediction-based resource monitoring and provisioning methods using neural networks is proposed in [7] along with linear regression to achieve on-demand resource allocation in the cloud, experimental results show that the suggested technique allows more flexible resource management for applications hosted in the cloud environment. The prediction models obtain a prediction accuracy of more than 80% for the workload being examined. Commonly used parametric models are the ARIMA[8] and the Kalman filter [9] based methods for data smoothing. In order to create a prediction module for Software as a Service (SaaS) providers, the authors [4] presents an ARIMA-based model specially designed for predicting cloud computing workloads. It does this by using actual web server request data. EEMD-RT-ARIMA[10] is another suggested hybrid technique to deal with non-stationary host utilization data. The data is divided into components of the Intrinsic Mode Function (IMF) and a residual component. The final forecast is then derived using an ARIMA prediction for each component. Furthermore, to improve the efficiency of decision trees, k-nearest neighbors, and support vector machine ensembles in workload prediction tasks in cloud computing settings, an intelligent ensemble approach [11] combines accuracy with a relative error-based pruning mechanism.

2.2 Workload prediction using Deep Learning Models

A self-directed workload forecasting method mainly focuses on web server traces such as NASA weblogs, Calgary Server, GCP trace, and Planet Lab trace. This method demonstrates a substantial reduction in mean squared forecast error, up to 99.99%, compared to existing techniques. It achieves this by enhancing the feedback window for forecast error, optimizing cluster size, and adjusting the learning rate. The method shows promise for further development and refinement in workload forecasting techniques[12]. A system for predicting workload based on Alibaba and Google Cloud Platform (GCP) traces has been developed using a Deep Neural Network (DNN) that incorporates a Gated Recurrent Unit (GRU) called es-DNN. Compared to state-of-the-art baselines, esDNN approach minimize mean square error leading to a 15% reduction compared to GRU alone. This predictive model can be extended to a container-based prototype system, making it adaptable and scalable. Additionally, the system's applicability extends to edge computing, where tasks can be offloaded to reduce response time, showcasing its versatility and efficiency in diverse computing environments. It is aded to reduce response time, showcasing its versatility and efficiency in diverse computing environments[13]. A workload prediction system leveraging Tencent and Alibaba Cloud using a Federated Learning (FL) setup aims to predict resource usage based on collected traces. The model incorporates an ensemble learning approach for workload prediction. The initial model utilizes Long Short-Term Memory (LSTM), and multiple global models are created using clustering techniques. Notably, the system is initially built using only two cloud traces, originating from

Tencent and Alibaba Cloud. However, it is designed to be extensible, allowing for the incorporation of multiple cloud traces. This flexibility enhances the model's generalizability and adaptability across diverse cloud environments, showcasing its potential scalability and effectiveness in predicting resource utilization across various platforms [14]. The authors of [15] focuses on workload prediction and container migration within the Alibaba Cluster trace. The developed Pro-Active Container migration at the Cloud employs an offline prediction model. However, the model's performance, as indicated by the achieved error rate, is noted to be considerably high. A significant limitation is that the research needed to address the methodology for selecting pods for migration. This absence of consideration for pod selection introduces a notable gap in the study's approach to pro-active container migration, suggesting further investigation and refinement in optimizing the migration strategy for improved efficiency . The work in [16] discusses workload prediction using weblogs from Complutense University, Madrid, and employs Long Short-Term Memory (LSTM) as the sole predictive model. The evaluation metrics include Root Mean Square Error (RMSE), Mean Square Error (MSE), and Mean Absolute Error (MAE). The LSTM model's prediction accuracy is reported with values of 0.043 for MAE, 0.075 for RMSE, and 0.066 for MSE. They proposed the LSTM model and other models for predicting resource requirements. The LSTM model is specifically highlighted for its application in hybrid auto-scaling. The authors of [17] focus on resource management using the Alibaba dataset and employs K-means clustering and a heuristic approach to minimize energy consumption. The results indicate a significant reduction in energy consumption, with 1.18 times and 2.35 times less energy utilized through the applied strategies. The proposed approach can be extended by offloading tasks and containerized applications from fog to the data center, contributing to further energy minimization. This extension underscores the adaptability of the methodology to broader computing environments, emphasizing the potential for efficient resource management and energy conservation by relocating tasks and applications between fog and data center infrastructure.

Table 1: Summary of previous works on workload prediction

Strategy	works	Methodology	Dataset	Target Resource
Statistical	[4]	ARIMA	Wikimedia web requests	HTTP requests
	[6]	ARIMA+ANN	Google Cluster Trace Bitbrains	CPU
Deep Learning	[18]	BI-LSTM,GridLSTM	Google Cluster Trace	CPU, RAM
	[19]	TCN,GRU	Bitbrains	CPU
	[20]	Attention based LSTM	Alibaba, Dinda	CPU
	[21]	CNN-LSTM	Bitbrains	CPU,Memory
	[22]	LSTM	Google Cluster Trace	CPU
	[23]	CNN+GRU	Google Cluster Trace	CPU
	[24]	VTGAN	Planet Lab	CPU
	[25]	PSO-GA, CNN-LSTM	Google Cluster Trace	CPU
	[26]	BI-LSTM	Bitbrains	CPU
	[27]	esDNN,GRU	Alibaba	CPU
	[28]	DCRNN	Planet Lab	CPU
	[29]	LSTM	Alibaba, Tencent	CPU
	[30]	CNN+BILSTM	Alibaba	CPU
Our Approach		KF-CLA	Alibaba	CPU,Memory
		KF-CBiLA	Materna	CPU,Memory
		KF-CGA	Bitbrains	CPU
			Microsoft Azure	CPU
			Planet Lab	CPU

3 Problem Description

Infrastructure providers are actively transforming transit and urban mobility as cloud, edge, and Internet of Things (IoT) technologies spread. They distribute sensors throughout road networks and metro lines to gather real-time data on environmental elements, vehicle movements, and traffic conditions. The latency-sensitive feature of autonomous automobile operations is actively addressed by edge data centers strategically positioned along main routes and near metro stations. Figure 1 represent the Edge data centers in metro scenario. These centers handle incoming sensor data in real time, actively reducing propagation delays and enabling timely decision-making for autonomous vehicles. Edge data centers aggressively prepare for resource overload during high-traffic or dynamic events. They anticipate workloads and evaluate the availability of resources in advance to proactively handle this difficulty. If a possible overflow is expected, they take preemptive steps, such as optimizing current resources or temporarily ramping up capacity. The issue of managing resources at edge servers to effectively manage the infrastructure resources for autonomous cars is relevant in this context.

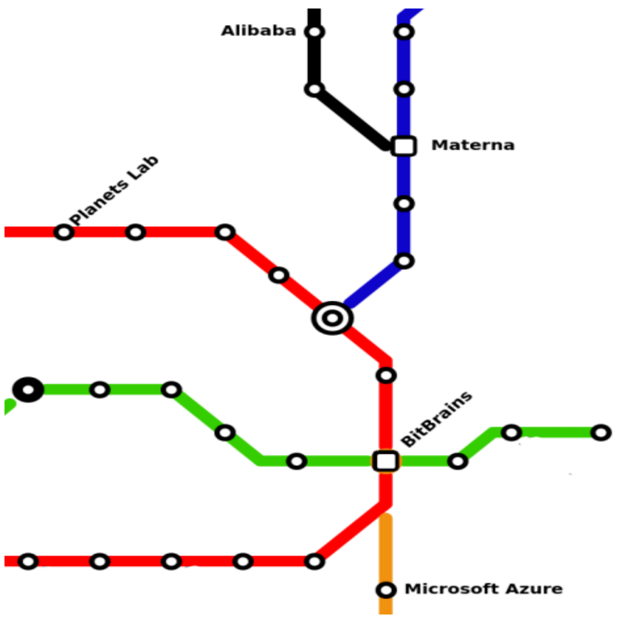


Fig. 1: Smart-City Metro Scenario

Let W represent a set of workload values over time t, where w_t denotes the workload at time t. Additionally, let h be a function that analyzes previous workload instances to estimate the outcome of future events, expressed as:

$$\hat{w}_{t+1} = h(w_t, w_{t-1}, ..., w_1) \tag{1}$$

the equations for the provided workload values, predicted values, and error calculation are:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(w_i - \hat{w}_i)^2 \tag{2}$$

For example, Workload values:

$$W = \{12.3, 14.7, 16.5, 18.2, 20.1, 22.6, 25.3, 27.8, 30.4, 33.1\} \tag{3}$$

Predicted values:

$$\hat{W} = \{12.4, 13.1, 15.5, 17.8, 19.9, 22.3, 24.8, 27.1, 30.2, 32.8, 35.5\} \tag{4}$$

Error calculation:

$$E = \{0.1, -0.8, -1.0, 0.4, -1.0, -0.3, 0.5, -0.7, 0.2, 0.3, -0.4\} \tag{5}$$

Let w_t represent the workload at time t, and let W symbolize a set of workload values across time t. Similarly, $\hat{W}$ represents a set of expected workload values for each time instance. A forecasting model based on historical workload cases calculates these projections. The forecast errors, represented by E, are computed as the discrepancies between the actual and expected workload values at each time.

The values W, $\hat{W}$, and E are randomly created. They present a fictitious scenario in which there could be a difference between the expected and actual workload values, causing prediction errors to fluctuate over time.

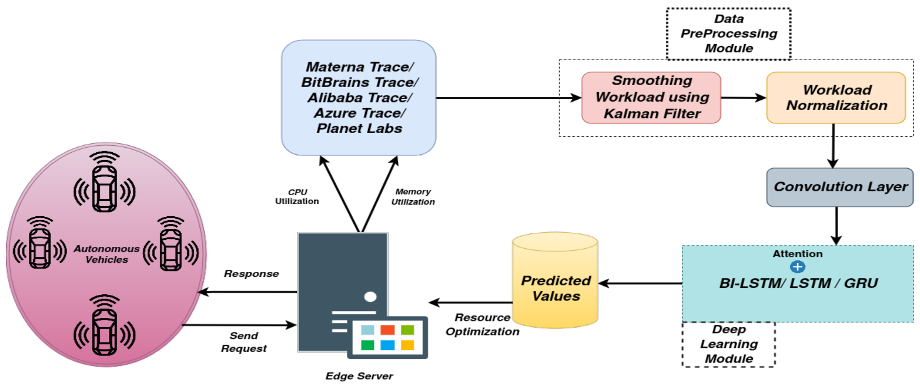


Fig. 2: Proposed Workload Prediction System

4 Proposed Approach

In our proposed approach, the raw multivariate time series data is pre-processed using a Kalman filter to smooth the data; it effectively removes the noise and outliers from the input sequence. The CNN component typically extracts the local features by applying the convolution layer, which captures patterns like seasonality, allowing the model to learn representations of data invariant to small shifts or variations in time. Figure 2 represents the proposed workload prediction system. LSTM components capture complex long-term dependencies and temporal information in time series data. It remembers information over long time intervals and selectively forgets irrelevant information. By using encoded representations from the CNN and LSTM, BI-LSTM and GRU layers to dynamically generate weights, the attention mechanism enables the model to concentrate on pertinent input segments. This adaptive allocation increases task relevance and enhances interpretability and performance by prioritizing informative characteristics or time steps. The Bidirectional LSTM component records both forward and backward dependencies to improve temporal comprehension. Figure 3 shows the CNN with attention model. Compared to LSTM, the GRU component has a more straightforward design yet captures temporal relationships. The attention mechanism dynamically computes weights based on encoded representations from the LSTM, BI-LSTM and GRU layers to enable the model to concentrate on pertinent input components. This adaptive allocation increases task relevance and enhances interpretability and performance by prioritizing informative characteristics or time steps.

Table 2: Baselines(WoK) and proposed models(WK)

Baseline Models	**Description**
CLA	Convolutional Long Short Term Memory with Attention
CBiLA	Convolutional Bi-directional Long Short Term Memory with Attention
CGA	Convolutional Gated Recurrent Unit Attention
ARIMA	Auto Regressive Integrated Moving Average
Proposed Models	
KF-CLA	Kalman filter and Convolutional Neural Network based LSTM with Attention
KF-CBiLA	Kalman filter and Convolutional Neural Network based Bi-directional LSTM with Attention
KF-CGA	Kalman filter and Convolutional Neural Network based Gated Recurrent Unit with Attention

4.1 Kalman Filter

In our proposed approch the raw data is passed to Kalman filter, it process the data in four different steps.

- $\hat{x}_{k|k-1}$ is the predicted state estimate at time k given observations up to time $k-1$.
- $P_{k|k-1}$ is the predicted state covariance matrix at time k given observations up to time $k-1$.
- F_k is the state transition matrix.
- B_k is the control-input matrix.

- u_k is the control input at time k.
- Q_k is the process noise covariance matrix.
- $\hat{z}_k$ is the predicted measurement.
- S_k is the predicted measurement covariance matrix.
- H_k is the measurement matrix.
- R_k is the measurement noise covariance matrix.
- K_k is the Kalman gain.
- z_k is the actual measurement at time k.
- I is the identity matrix.

4.1.1 State Prediction:

$\hat{x}k|k-1$, the state estimate at time k, is predicted using the following information: the previous state estimate $\hat{x}k-1|k-1$; the state transition matrix F_k represents the dynamics of the system; and the control input u_k, if it is available, incorporating B_k. The expected covariance matrix $P_{k|k-1}$ accounts for the uncertainty in this prediction. It is calculated by adding the process noise covariance Q_k and propagating the uncertainty from the previous time step using F_k. The state prediction equations are:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \tag{6}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{7}$$

4.1.2 Measurement Prediction:

The expected measurement at time k is indicated by $\hat{z}k$, which is calculated using the measurement matrix H_k and the projected state $\hat{x}k|k-1$. This prediction's uncertainty is represented by the covariance matrix S_k. It is computed by integrating the measurement noise covariance R_k and changing the state uncertainty with the help of the measurement matrix H_k. The measurement prediction equations are:

$$\hat{z}_k = H_k \hat{x}_{k|k-1} \tag{8}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{9}$$

4.1.3 Kalman Gain Calculation:

Based on the difference between the actual measurement z_k and the projected measurement $\hat{z}_k$, the Kalman gain K_k at time k determines the adjustment made to the predicted state estimation. It is computed as the inverse of the sum of the prediction and measurement covariances, the transpose of the measurement matrix H_k^T, and the anticipated state covariance $P_{k|k-1}$. The Kalman gain calculation equation is:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{10}$$

4.1.4 State Update:

Real measurement z_k is included to update the state estimate to $\hat{x}_{k|k}$. The update is obtained by applying the Kalman gain K_k to the projected state estimate $\hat{x}_{k|k-1}$. The uncertainty that results from using the actual measurement is reflected in the revised covariance matrix $P_{k|k}$. It is calculated by applying the Kalman gain to the expected state covariance $P_{k|k-1}$. The state update equations are:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - \hat{z}_k) \tag{11}$$

$$P_{k|k} = (I - K_k H_k)P_{k|k-1} \tag{12}$$

4.2 Normalization

After Smoothing the data using Kalman Filter, next step is to scale the dataset's features to a specific range typically between 0 and 1 which is the aim of normalisation. By preventing features with greater scales from controlling the learning process, this procedure guarantees that each feature contributes equally to the analysis. In our approach Min-Max scaling is adopted. Min-Max scaling, which scales the data to a predetermined range depending on the minimum and maximum values of each feature, is a popular technique for data normalisation. Min-Max scaling is especially helpful in situations when the feature distribution is non-Gaussian or uncertain. Min-Max Scaling has thus been applied to all of the data traces. The Min-Max scaling formula for a feature x is given by:

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{13}$$

Where:

- x_{scaled} is the scaled value of the feature x.
- $\min(x)$ is the minimum value of the feature x in the dataset.
- $\max(x)$ is the maximum value of the feature x in the dataset.

4.3 1D Convolutional Neural Network

In time series forecasting architectures, there are several benefits to integrating a 1D convolutional layer before recurrent layers. Convolutional layers are great for extracting local patterns' characteristics, such as trend and seasonality. Down sampling methods help reduce dimensionality by concentrating on essential features and minimizing computing burden. The recurrent layers capture broader temporal patterns, such as long-term dependencies, while the Conv1D layer focuses on extracting finer details and local temporal features. Furthermore, the model learns abstract characteristics resilient to temporal shifts or fluctuations in the data. It improves generalization and strengthens the model's capacity to understand complicated relationships in the time series data that involve long-term dependencies. This results in higher forecasting accuracy but requires empirical validation to determine the best practices for particular datasets and tasks.

4.4 Attention Mechanism

Attention Mechanism: The attention mechanism is a critical component in sequence modeling tasks, enabling models to focus on relevant parts of the input sequence. It dynamically learns to assign different weights to each time step's information, allowing the model to capture dependencies and make accurate predictions. Suppose we have a sequence of hidden states from an LSTM/Bi-LSTM/GRU layer:

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ \vdots & \vdots & \vdots \\ h_{T,1} & h_{T,2} & h_{T,3} \end{bmatrix}$$

where T is the number of time steps, and each h_t represents the hidden state vector at time step t

Dense Layer: The output from the LSTM/Bi-LSTM/GRU layer is passed through a dense layer with a single unit and ReLU activation, producing a vector D. This layer learns to capture the importance of each hidden state.

$$D = \text{ReLU}(W_d \cdot H + b_d)$$

where W_d and b_d are the weight matrix and bias vector of the dense layer, respectively.

Suppose we have H as described earlier. After passing it through the dense layer, we obtain:

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_T \end{bmatrix}$$

Softmax Activation: The vector D is then passed through a softmax activation function after flattening to obtain attention weights a. Softmax normalizes the values to ensure they sum up to 1, representing probabilities.

$$a_i = \frac{e^{d_i}}{\sum_{i=1}^{T} e^{d_i}} \tag{14}$$

where d_t is the t^{th} element of D. Suppose D = [0.1, 0.2, 0.3]. After applying softmax, we get a = [0.267, 0.367, 0.366].

Repeat Vector: The attention weights a obtained from softmax are then repeated T times to match the dimensions of H. This step ensures that each weight corresponds to a respective hidden state.

$$A = Repeat(a, T) \tag{15}$$

If a = [0.267, 0.367, 0.366], after repeating, we get:

$$A = \begin{bmatrix} 0.267 & 0.367 & 0.366 \\ 0.267 & 0.367 & 0.366 \\ \vdots & \vdots & \vdots \\ 0.267 & 0.367 & 0.366 \end{bmatrix}$$

Permute Layer: After repeating the attention weights a to match the number of time steps in the LSTM/Bi-LSTM/GRU outputs H, the Permute layer is employed to adjust the shape of the repeated attention weights. This adjustment ensures that each weight corresponds to the respective hidden state, enabling accurate weighting of the information from each time step. Suppose we have T time steps in the input sequence and T corresponding hidden states from the LSTM/Bi-LSTM/GRU layer. After repeating the attention weights a, we obtain a matrix A of shape (T, 1), where each row represents the repeated attention weights for a specific time step. However, for proper element-wise multiplication with the LSTM/Bi-LSTM/GRU outputs H, it is essential to align the dimensions of A with H. The Permute layer performs this operation by rearranging the dimensions of A to match the shape of H. Suppose A is repeated attention weights with shape (T, 1), and H is LSTM/Bi-LSTM/GRU outputs with shape (T, N), where N is the dimensionality of the hidden states. The Permute layer rearranges the dimensions of A to (1, T), aligning it with the shape of H. The Permute layer ensures that each attention weight is correctly associated with the corresponding hidden state, facilitating accurate weighting of the information from each time step. This alignment is crucial for the attention mechanism to effectively capture the relevance of each time step's information in the input sequence. By employing the Permute layer, the attention mechanism ensures proper alignment of attention weights with hidden states, enabling the model to focus on relevant parts of the input sequence. This alignment enhances the model's ability to capture dependencies and make accurate predictions.

Multiply Layer: The Multiply layer is a fundamental component in the attention mechanism, responsible for performing element-wise multiplication between the adjusted attention weights and the LSTM/Bi-LSTM/GRU outputs. This operation emphasizes the importance of each hidden state based on the corresponding attention weight. After the attention weights are repeated and permuted to align with the dimensions of the LSTM/Bi-LSTM/GRU outputs, the Multiply layer carries out element-wise multiplication between the adjusted attention weights and the 2hidden states. This operation allows the model to assign higher importance to hidden states with higher attention weights, effectively focusing on the most relevant parts of the input sequence. Suppose we have adjusted attention weights A and LSTM/Bi-LSTM/GRU outputs H, where each row of A corresponds to the attention weights for a specific time step. The Multiply layer computes element-wise multiplication between each row of A and the corresponding row of H, emphasizing

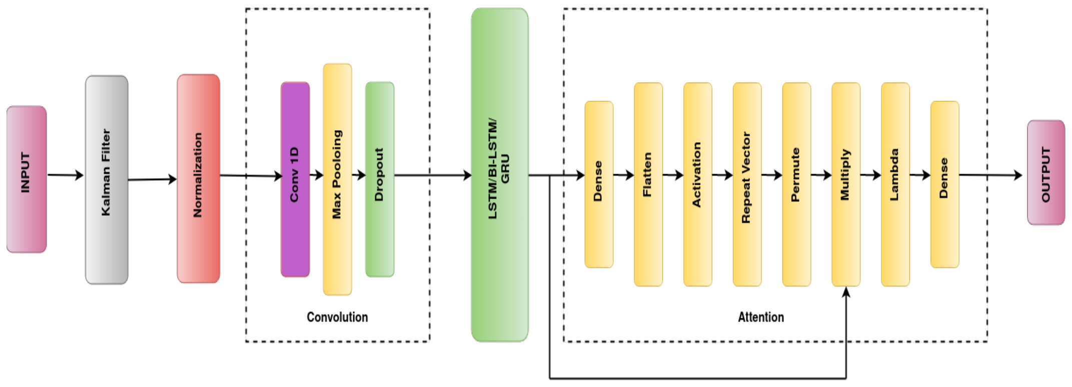


Fig. 3: CNN-Attention Model

the importance of each hidden state based on its attention weight. The Multiply layer plays a crucial role in the attention mechanism by allowing the model to selectively emphasize the importance of each hidden state in the input sequence. By performing element-wise multiplication, the Multiply layer ensures that the model focuses on relevant parts of the sequence, facilitating accurate computation of the context vector.
Lambda Layer: The Lambda layer is used to compute the weighted sum of the element-wise multiplied values obtained from the Multiply layer. This operation aggregates the information from all time steps, producing the final context vector that captures the most relevant information from the input sequence. After element-wise multiplication is performed between the adjusted attention weights and the LSTM/Bi-LSTM/GRU outputs, the Lambda layer computes the weighted sum of the resulting vectors. This operation aggregates the weighted information from all time steps, producing a context vector that represents the most relevant information from the input sequence. Suppose we have computed the element-wise multiplication between adjusted attention weights A and LSTM/Bi-LSTM/GRU outputs H, resulting in a set of weighted vectors. The Lambda layer sums up these weighted vectors across all time steps, yielding the final context vector that captures the most important information from the input sequence. The Lambda layer serves as the final step in the attention mechanism, aggregating the information from all time steps to produce a context vector that encapsulates the most relevant information from the input sequence. By computing the weighted sum, the Lambda layer enables the model to focus on important parts of the sequence, facilitating accurate prediction and decision-making. In summary, the multiply layer performs element-wise multiplication to emphasize the importance of each hidden state based on the corresponding attention weight, while the Lambda layer computes the weighted sum of the resulting vectors to produce the final context vector. Together, these layers enable the attention mechanism to focus on relevant parts of the input sequence, enhancing the model's performance in sequence modeling tasks.

4.5 LSTM

The LSTM networks are comprised of LSTM cells, each containing a cell state and several gates that regulate information flow. These gates include the forget gate, input gate, and output gate. The essential components of an LSTM cell are: Cell State (C_t): This represents the long-term memory of the cell and retains information over extended sequences. Forget Gate (f_t): Responsible for determining which information should be discarded from the cell state. Input Gate (i_t): Controls the selection of new information to be added to the cell state. Output Gate (o_t): Governs the flow of information from the cell state to the output. In time series prediction tasks like forecasting CPU utilization, memory utilization historical data points of the time series (e.g., CPU and memory usage over time) serve as input features for predicting future values. Figure 4 represents LSTM model. LSTM networks excel in capturing temporal dependencies within input data, including seasonal patterns, trends, and periodic behaviors. This ability is crucial for achieving accurate predictions in time series analysis.

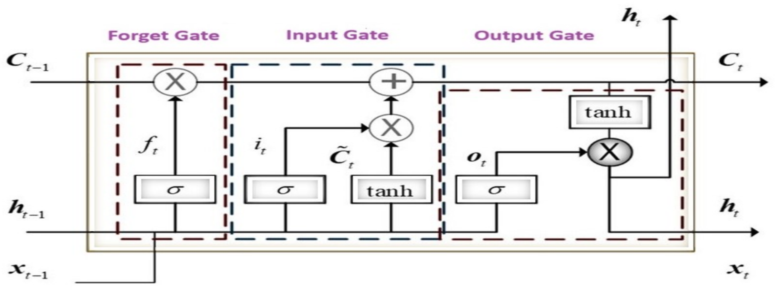


Fig. 4: LSTM Model

4.6 BI-LSTM

Bidirectional Long Short-Term Memory (Bi-LSTM) is an extension of the traditional LSTM architecture that incorporates information from both past and future time steps. Figure 5 represents Bi-LSTM model. Bi-LSTM networks are powerful for capturing bidirectional dependencies in sequential data, making them well-suited for time series-based prediction tasks such as CPU utilization and memory utilization prediction. Bi-LSTM networks consist of two LSTM layers: a forward LSTM layer and a backward LSTM layer. Each LSTM layer processes the input sequence independently, with the forward LSTM layer processing the sequence in the forward direction , and the backward LSTM layer processing it in the backward direction. The outputs of both LSTM layers are concatenated at each time step to capture bidirectional dependencies in the input sequence.

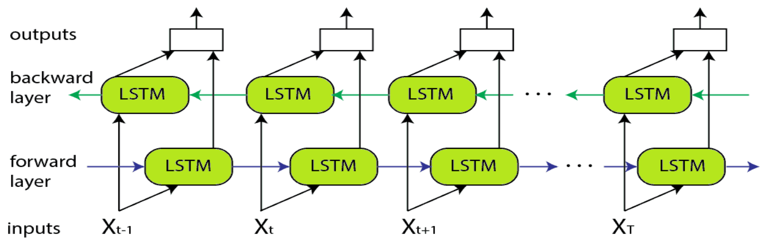


Fig. 5: BI-LSTM Model

4.7 GRU

GRU consists of a single recurrent layer with units known as GRU cells. Figure 6 represents GRU architecture. Each cell contains: Update Gate (z_t): Determines how much of the past information should be kept and how much of the new information should be added to the current state. Reset Gate (r_t): Controls which past information should be forgotten or reset. Current Memory (h_t): Represents the current state or memory of the cell, which is updated based on the input and gates.

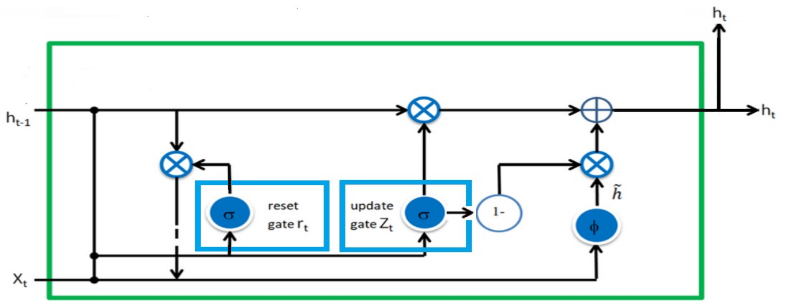


Fig. 6: GRU Model

4.8 ARIMA

Autoregressive Integrated Moving Average (ARIMA) is a widely used statistical method for time series forecasting, including tasks like CPU utilization prediction. Autoregressive (AR): Linear regression of the series against its lagged values. Integrated (I): Differencing the series to make it stationary (remove trends). Moving Average (MA): Linear combination of past error terms. The general equation for an ARIMA model of order (p, d, q) can be represented as follows:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \epsilon_t \tag{16}$$

Where:

- Y_t: Value of the time series at time t.
- c: Constant term.
- $\phi_1, \phi_2, \ldots, \phi_p$: Auto regressive coefficients.
- $\theta_1, \theta_2, \ldots, \theta_q$: Moving average coefficients.
- ϵ_t: Residual error at time t.

Time series data from the past is used to train ARIMA models. The model's parameters (p, d, and q) are chosen using methods like grid search or information criteria like the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC). ARIMA models can be used to forecast future time steps once they have been trained. Forecasts are produced by the model using the observed values and previously determined parameters.

Table 3: Parameters of Proposed Model

Parameters	**Values**
Units in Recurrent Layers	50
Batch_size	16
Window_size	1,5,10,15,30,60
Activation Function of Attention Layer	ReLU
Number of Convoluational layers	1
Activation Function of Convolution Layers	ReLU
Convolution Layer Filters	64
Convolution Layer Kernel Size	3
Train Ratio	0.4
Validation Ratio	0.2
Test Ratio	0.4
Dropout	0.2
Epochs	100
Loss Function	Mean Squared Error (MSE)
Additional Metrics	Mean Absolute Error (MAE)
Optimizer	Adam
Learning Rate	0.001

5 Time Complexity

The time complexity of data preparation, which involves scaling the data using MinMaxScaler, is around O(n), a function of the dataset size. The time complexity of producing training and testing data is O(n), where n is the dataset's length due to the iterative nature of the process. The LSTM/BI-LSTM/GRU model must be trained by making forward and backward runs through the network. This process has a time complexity of around O(N * T * D^2), where N is the total number of training sequences, T is the sequence length (window size), and D is the total number of hidden units in the LSTM/BI-LSTM/GRU layer. Finally, with a time complexity of about O(M * T * D), where M is the number of testing sequences, predicting using the trained model requires forward runs over the network. Considering these, the model's predicted time complexity is O(N * T * D^2).

5.1 Dataset Description

For our experiments we have evaluated proposed model on five different real time datasets and there details are as follows: The dataset repository includes a variety of traces gathered from many sources, each providing distinct information on the use of virtual machines (VMs) in varied environments. Over ten days in March and April 2011, the Planet Lab trace records the average CPU consumption collected from over 1000 virtual machines spread over roughly 500 worldwide locations. The Bitbrains TUDelft dataset consists of two subsets concentrating on virtual machine (VM) allocation at a remote data center. The first subset has a mixed population, and the second subset uses SAN storage and includes significant data on CPU, memory, disk, and network usage over several weeks. Furthermore, performance metrics from a dispersed data center encompassing three datasets totaling 1594 virtual machines collected over three months are included in the Materna trace. To enhance the dataset, we extracted the 'CPU cores' information from the 'Trace info' file and added a new column to each of the 520 CSV trace files, named 'cpu cores,' where we appended the corresponding CPU core values. Subsequently, we utilized these modified CSV files to train our model for multivariate data analysis, considering both 'CPU utilization percentage' and 'CPU cores' as features. The Alibaba cluster-trace-v2018 dataset contains precise information on container resource use and recordings from about 4000 computers over eight days. we extracted data pertaining to a single machine ID, 'm_1717,' selected randomly for our experiments. This resulted in a dataset comprising 39000 rows specific to this machine ID. Microsoft Azure dataset comprises multiple CSV files containing trace information such as 'hashed Machine ID,' 'average, minimum, maximum CPU utilization percentage,' and more. Additionally, there exists another file named 'vmtable,' which holds supplementary details like 'CPU cores' for machine IDs (in hashed form). To enhance the dataset, we executed a script to retrieve 'CPU cores' for the provided machine IDs, matching them with the original CSV files. Subsequently, we added a new column for 'CPU cores' in the CSV files and populated it with values sourced from the 'vmtable' file. Similar to the approach with the Matrena Trace dataset, we leveraged these modified CSV files to train our model for multivariate data analysis, incorporating both 'Average CPU utilization percentage' and 'CPU cores' as features.

5.2 Experimental Setup

The entire experiment was conducted on Google Colab Pro. The environment was configured with Python version 3.10.12. Key libraries utilized included Keras version 2.15.0, NumPy version 1.25.2, Pandas version 2.0.3, Matplotlib version 3.7.1, and PyKalman version 0.9.7. These tools provided the necessary framework for executing the experiment and analyzing the results effectively. The Alibaba cluster-trace-v2018 dataset, encompassing around 4000 machines observed over an 8-day span, we extracted data pertaining to a single machine ID, 'm_1717,' selected randomly for our experiments. This resulted in a dataset comprising 39000 rows specific to this machine ID. Due to the large dataset's size, we ran the extraction script on a system equipped with an NVIDIA GV100GL (Tesla V100 SXM2 32GB) GPU, clock @ 33MHz. The source code of the models and graphs can be found at github [31].

5.3 Evaluation Metrics

The performance of the proposed hybrid model is evaluated using Mean Squared Error (MSE) and the equation for calculating it is given by equation 17, and Mean Absolute Error (MAE) is calculated using equation 18 and they are the significant performance indicators. The output values (y_i) and the matching projected values $(\hat{y}_i)$ are used to construct these measures.

Additionally, the MSE values show minimum variance for both the training and test data sets, suggesting that the model's generalization is adequate. This shows that the resilience and reliability of the models are enhanced by its persistent good performance across many datasets.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{17}$$

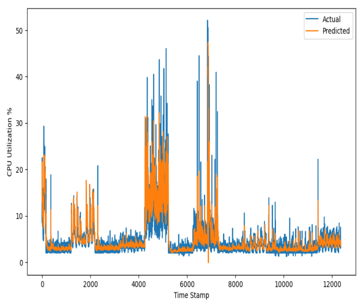
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{18}$$

Table 4: Comparison of proposed models with existing works considering CPU % utilization as a feature

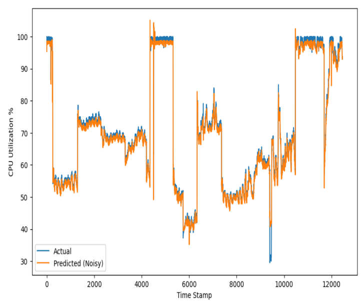
Model	Dataset	MSE	MAE	RMSE
SG-CBA[30]	Alibaba	0.00104	0.023914	0.03234
GRU[29]		0.36567	0.51289	0.6047
LSTM[20]		0.04134	0.03520	0.20330
CNN+LSTM[21]	Bitbrains	0.299769	0.137823	0.03765
Bi-LSTM[26]		66.80	5.19	8.17
ALEDAR[32]		17.98	-	4.240
VTGAN(GRU-Based)[24]	Planet Lab	0.94772	0.705	0.873
DCRNN[28]		5.947	-	2.438
MLP-ANN[33]	Materna	153.76	-	12.40
KF-CGA (Proposed)	Alibaba	**0.00101**	**0.01205**	**0.03177**
KF-CGA (Proposed)	Bitbrains	**0.00561**	**0.06583**	**0.07490**
KF-CGA (Proposed)	Planet Lab	**0.01801**	**0.10054**	**0.13400**
KF-CGA(Proposed)	Materna	**0.00319**	**0.04875**	**0.05645**

Table 5: Alibaba_v2018 trace MSE and MAE for CPU utilization % and Memory utilization % using different prediction models

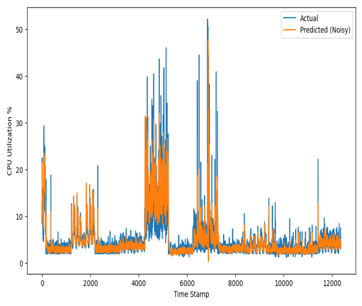
Model	PWS	CPU(WK)		Memory(WK)		CPU(WoK)		Memory(WoK)		% Decrease	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
KF-CLA	1	0.00190	0.02355	**0.00066**	**0.01617**	0.01515	0.08784	0.09140	0.01992	85.91814	68.30194
	5	**0.00162**	**0.02005**	0.00088	0.01881	0.08957	0.13731	0.09745	0.09923	**97.75002**	83.66033
	10	0.00423	0.04131	0.00120	0.01887	0.01436	0.20340	0.03174	0.09325	34.04231	74.22018
	30	0.00182	0.02220	0.00176	0.09912	0.07634	0.09954	0.29103	0.31843	67.79151	16.32977
	60	0.17830	0.36793	0.08815	0.21343	0.78436	0.93234	0.61285	0.86735	77.07288	**85.81885**
KF-CBiLA	1	0.00152	**0.01822**	**0.00058**	0.01564	0.08357	0.10435	0.09187	0.17478	68.61419	53.96164
	5	**0.00140**	0.02001	0.00084	**0.01412**	0.09803	0.29437	0.08676	0.10829	17.89146	**90.54795**
	10	0.00420	0.04082	0.00101	0.01717	0.0403	0.37326	0.01436	0.08732	**82.30852**	87.05269
	30	0.00166	0.02147	0.00144	0.09984	0.1704	0.65341	0.05657	0.30543	52.99137	84.95194
	60	0.14725	0.37236	0.06275	0.18815	0.81329	0.98327	0.21436	0.64365	62.71368	21.30533
KF-CGA	1	**0.00101**	**0.01205**	0.00051	0.01440	0.90810	0.03999	0.08343	0.13543	**90.71116**	24.40053
	5	0.00120	0.01827	**0.00041**	**0.01108**	0.10435	0.09832	0.06769	0.07982	16.59752	**48.79775**
	10	0.00205	0.03008	0.00075	0.01209	0.0702	0.07703	0.08731	0.08897	86.99693	34.01162
	30	0.00151	0.02136	0.00124	0.09954	0.10345	0.10523	0.09932	0.53253	12.67314	40.95481
	60	0.10517	0.29843	0.05966	0.18592	0.67802	0.39983	0.45475	0.85546	76.96396	27.43965
KF-ARIMA		6.73267	5.64753	5.36482	2.64873	12.65437	8.65624	11.65768	5.63463	46.79569	34.51702



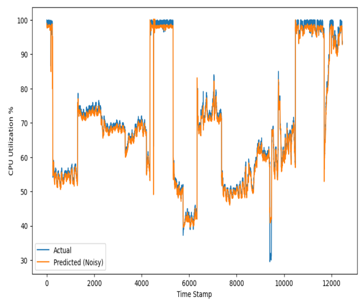
(a) LSTM-Cpu_Util% PWS=5mins



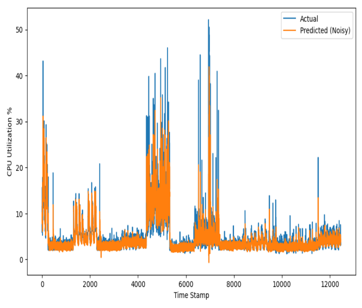
(b) LSTM-Memory_Util% PWS=1min



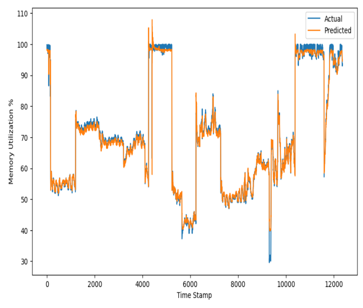
(c) BI-LSTM Cpu_Util % PWS=5mins



(d) BI-LSTM Memory_Util% PWS=1min



(e) GRU Cpu_Util% PWS=1min

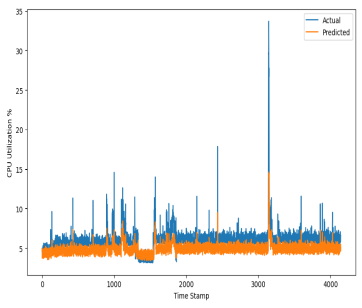


(f) GRU Memory_Util% PWS=5mins

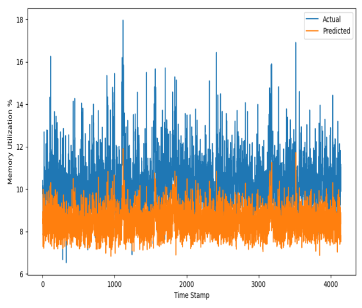
Fig. 7: Best results from Table 5 CPU and Memory utilization % of Alibaba Dataset

Table 6: Materna Trace MSE and MAE for CPU utilization % and Memory utilization % using different prediction models

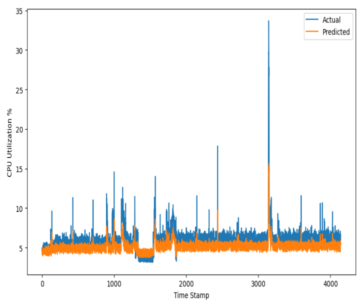
Model	**PWS**	**CPU(WK)**		**Memory(WK)**		**CPU(WoK)**		**Memory(WoK)**		**% Decrease**	
		MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
KF-CLA	10	0.00454	0.06132	0.03855	0.16316	0.06876	0.08132	0.23534	0.88769	70.51698	88.96718
	15	0.00379	0.06123	0.02994	0.15835	0.05234	0.08154	0.25983	0.88784	**73.45631**	**89.79195**
	30	0.00368	0.05822	**0.02685**	0.15774	0.06543	0.34839	0.10932	0.86635	41.13253	73.81022
	60	**0.00367**	**0.05712**	0.02703	**0.14875**	0.54667	0.43145	0.11234	0.89234	38.19092	78.26760
KF-CBiLA	10	0.00435	0.05732	0.02575	0.15702	0.02354	0.04435	0.23543	0.65632	**73.11889**	78.68608
	15	0.00379	0.05585	0.02481	0.14845	0.02679	0.04436	0.23454	0.66766	70.86908	78.93502
	30	0.00365	0.05564	**0.02214**	0.14778	0.02078	0.04613	0.22154	0.53254	69.87536	79.30678
	60	**0.00347**	**0.05397**	0.02312	**0.14367**	0.03435	0.06732	0.22635	0.51453	**73.10695**	**85.16973**
KF-CGA	10	0.00414	0.05188	0.02481	0.15255	0.05465	0.23543	0.24556	0.84325	**63.85212**	**62.77984**
	15	0.00371	0.05187	0.02421	0.14787	0.03432	0.23583	0.21345	0.84311	61.84992	62.75229
	30	0.00361	0.05123	0.02109	0.14563	0.09843	0.23432	0.21089	0.97326	63.22962	59.78364
	60	**0.00319**	**0.04875**	**0.01922**	**0.12752**	0.09993	0.23879	0.18978	0.65832	61.77403	59.11304
KF-ARIMA		3.42197	0.59733	2.46558	1.21530	23.34536	5.53425	11.32543	7.62832	32.21854	77.69660



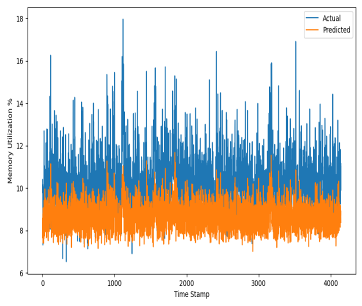
(a) LSTM-Cpu_Util% PWS=60mins



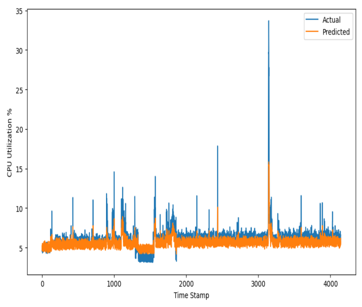
(b) LSTM-Memory_Util% PWS=30mins



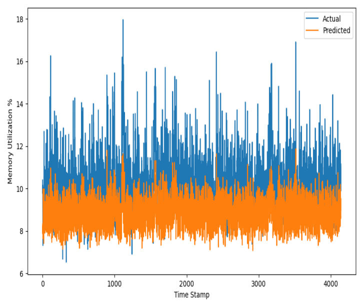
(c) BI-LSTM Cpu_Util % PWS=60mins



(d) BI-LSTM Memory_Util% PWS=30mins



(e) GRU Cpu_Util% PWS=60mins



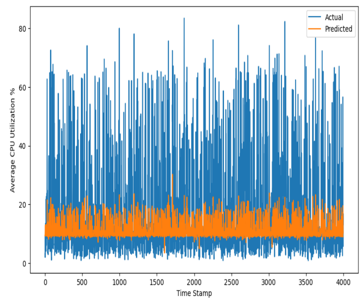
(f) GRU Memory_Util% PWS=60mins

Fig. 8: Best results from Table 6 CPU and Memory utilization % of Materna Dataset

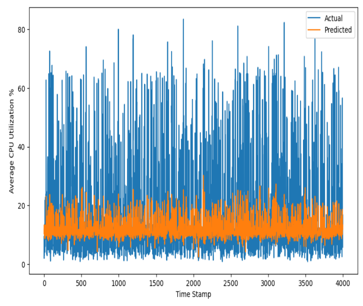
6 Results and Discussion

6.1 Alibaba_2018 Dataset

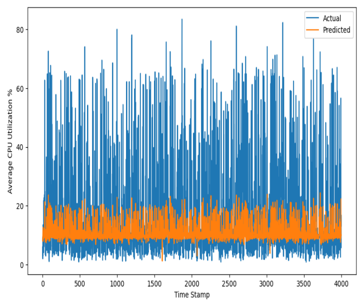
The models KF-CLA and KF-CBiLA perform similarly, reaching their optimal MSE and MAE values at a PWS of 5 minutes for CPU and memory utilization, respectively.



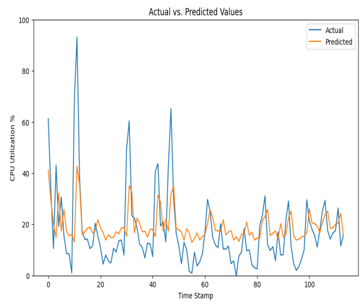
(a) LSTM-AvgCpu_Util% PWS=15mins



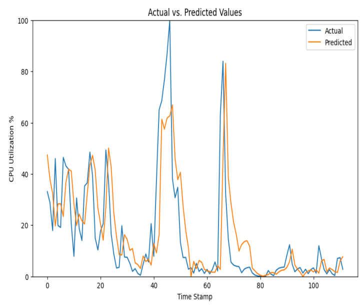
(b) BI-LSTM AvgCpu_Util% PWS=30mins



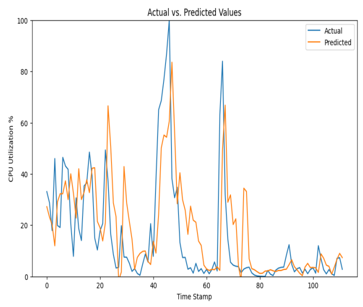
(c) GRU Avg_Cpu_Util% PWS=10mins



(d) LSTM Cpu_Util% PWS=10mins



(e) BI-LSTM AvgCpu_Util% PWS10mins



(f) GRU Avg_Cpu_Util% PWS=10mins

Fig. 9: Best results of Microsoft Azure dataset in sub figures (a), (b), (c) and Planet Lab Dataset in sub figures (d), (e), (f)

Table 7: Planet Lab Trace MSE and MAE for CPU utilization % using different prediction models

Model	**PWS**	**CPU(WK)**		**CPU(WoK)**		**% Decrease**	
		MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
KF-CLA	10	**0.01795**	**0.09897**	0.09783	0.36547	19.82728	60.80365
	15	0.01839	0.10028	0.28493	0.77634	**72.29619**	76.98215
	30	0.01968	0.10280	0.30238	0.88763	67.80852	**77.65151**
	60	0.01944	0.11004	0.30578	0.88673	70.98214	61.25736
KF-CBiLA	10	**0.01838**	0.09952	0.21897	0.61324	59.38329	70.11341
	15	0.01883	**0.09861**	0.24786	0.61357	**63.78425**	62.10162
	30	0.01914	0.10093	0.22546	0.87324	60.08788	**72.12088**
	60	0.01935	0.10287	0.23534	0.86254	61.18776	66.5726
KF-CGA	10	**0.01657**	**0.09074**	0.10278	0.23423	32.01649	15.70886
	15	0.01699	0.09473	0.12243	0.27382	35.51930	27.01805
	30	0.01776	0.09932	0.12548	0.73762	32.60151	**72.68842**
	60	0.01801	0.10054	0.14638	0.77861	**40.87908**	71.04388
KF-ARIMA		64.53023	4.71363	66.78234	55.47873	3.41407	58.78484

In contrast, KF-CGA achieves the lowest MSE and MAE values at a PWS of 1 minute and 5 minutes for memory utilization. Table 5 summarizes the three models MSE and MAE with respect to CPU and memory utilization % With Kalman filter and Without using Kalman filter. KF-CGA shows more resilience, keeping comparatively steady performance across a range of PWS values, showing its resistance to variations in the PWS, KF-CLA and KF-CBiLA show noticeable performance degradation at higher PWS values, with a decrease in prediction accuracy. While KF-CGA performs similarly to KF-CLA and KF-CBiLA but has a slightly different optimal PWS for CPU and memory utilization %, indicating varying strengths across different prediction models, KF-CLA and KF-CBiLA show similar performance trends overall, with KF-CLA marginally outperforming KF-CBiLA in specific scenarios. KF-CBiLA at 10 minutes PWS achieves 82% decrease in MSE and at 5 minutes PWS 90% decrease in MAE from the Baselines, KF-CGA at 1 minute and 5 minutes PWS achieves 90% and 48% decrease in MSE, respectively. KF-CLA at PWS 5 minutes achieves a 97% decrease in MSE and MAE at PWS 60 minutes, an 85% decrease in MAE from the Baselines. Overall, for the Alibaba dataset, our suggested models outperform all baselines and KF-CLA and KF-CBiLA at various PWS; however, as PWS increases, there is a little decline in performance. Figure 7 give the best results from Table 5 CPU and Memory utilization % of Alibaba Dataset [34].

Table 8: Microsoft_Azure_2019 Trace MSE and MAE for Average CPU utilization % using different prediction models

Model	**PWS**	**CPU(WK)**		**CPU(WoK)**		**% Decrease**	
		MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
KF-CLA	10	0.02234	**0.10112**	0.03894	0.46547	42.52137	57.39304
	15	**0.02230**	0.10187	0.49345	0.73271	**95.39282**	75.61913
	30	0.02249	0.10689	0.08734	0.80325	73.38713	**85.21713**
	60	0.02303	0.10911	0.09893	0.82143	55.15766	77.07087
KF-CBiLA	10	0.02223	**0.10176**	0.02893	0.87832	23.16904	83.64413
	15	0.02234	0.10598	0.08567	0.89372	73.92180	**83.68799**
	30	**0.02214**	0.10560	0.09892	0.64076	**77.61584**	72.44109
	60	0.02303	0.10854	0.08743	0.60832	73.65411	69.04283
KF-CGA	10	**0.02239**	**0.10073**	0.56547	0.26578	**96.03917**	61.15089
	15	0.02248	0.10884	0.23456	0.83458	90.41278	**86.40568**
	30	0.02264	0.10130	0.34776	0.83253	93.48809	81.42457
	60	0.02302	0.10674	0.37566	0.19832	93.87019	42.33900
KF-ARIMA		63.33605	10.60669	83.23432	16.54636	19.20949	28.34223

6.2 Materna Dataset

Identical performance patterns are seen in the KF-CLA, KF-CBiLA, and KF-CGA models. The best MSE and MAE values are obtained for CPU utilization percentage at a prediction window size (PWS) of 60 minutes. On the other hand, KF-CLA and KF-CBiLA perform better for memory use percentage at a PWS of 30 minutes for MSE. Interestingly, at a PWS of 60 minutes, KF-CLA and KF-CBiLA both produce improved MAE values for memory use percentage. In particular, KF-CLA reduces MSE by 73% and MAE by 89% for CPU use percentage at a 15-minute PWS. Similarly, at a PWS of 60 minutes for CPU usage for KF-CBiLA, there is a notable 73% drop in MSE and an 85% reduction in MAE. Furthermore, at a one-minute PWS, KF-CGA demonstrates a 62% reduction in MAE and a 63% reduction in MSE. These results imply that for the Materna trace dataset, the suggested models perform better at higher PWS and comparatively worse at lower PWS. Figure 8 give the best results from Table 6 CPU and Memory utilization % of Materna Dataset [35].

6.3 Planet lab Dataset

The MSE and MAE values for CPU utilization percentage and CPU(WoK) for several prediction models at different prediction window sizes (PWS) are displayed in the table. Notably, KF-CLA, KF-CBiLA, and KF-CGA all perform better than KF-ARIMA constantly. KF-CGA, in particular, has the lowest MSE and MAE values for PWS 10 minutes, especially at higher PWS, demonstrating its predictive power

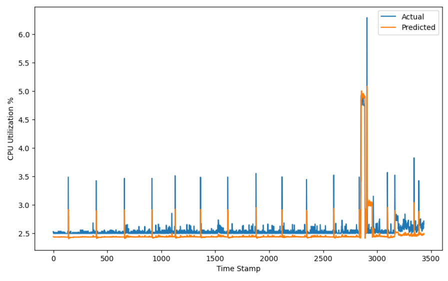
Table 9: Bitbrain's Trace MSE and MAE for CPU utilization % using different prediction models

Model	**PWS**	**CPU(WK)**		**CPU(WoK)**		**% Decrease**	
		MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
KF-CLA	1	**0.00544**	**0.06563**	0.08890	0.13253	**91.18881**	32.67017
	5	0.00546	0.06565	0.08322	0.35465	91.10975	**75.09599**
	15	0.00547	0.06567	0.04908	0.84389	88.11845	89.66289
	30	0.00561	0.06583	0.23466	0.80124	68.31533	89.03797
	60	0.00579	0.06627	0.25464	0.78325	67.21228	87.71618
KF-CBiLA	1	**0.00543**	**0.06561**	0.04575	0.32543	80.91467	73.13292
	5	0.00544	0.06565	0.045989	0.43543	80.78709	81.31863
	15	0.00545	0.06567	0.03987	0.35654	78.05808	75.05924
	30	0.00560	0.06582	0.04768	0.45676	82.36026	80.53140
	60	0.00574	0.06587	0.06893	0.54687	**87.44364**	**84.94258**
KF-CGA	1	**0.00533**	**0.06542**	0.03456	0.23543	74.00856	63.60477
	5	0.00540	0.06545	0.08435	0.32654	**91.02902**	**72.91798**
	15	0.00541	0.06561	0.07284	0.22154	86.63420	57.31484
	30	0.00551	0.06582	0.06345	0.32436	84.81954	71.16020
	60	0.00561	0.06583	0.05635	0.33367	82.90679	72.32485
KF-ARIMA		0.14564	0.10423	2.45436	4.54654	63.43955	72.85092

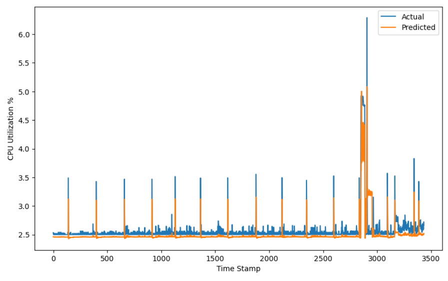
for CPU utilization percentage. MSE and MAE values generally fall as PWS rises for all models, indicating better predictive performance across longer prediction horizons. Although the MSE and MAE values of KF-CLA are generally higher than those of KF-CBiLA and KF-CGA at PWS 60 minutes, all suggested models outperform KF-ARIMA regarding prediction accuracy. These results highlight the significance of taking longer-term trends into account for precise CPU utilization forecasts, and the suggested models present viable substitutes for the baseline model. KF-CBiLA at 15 minutes PWS achieves 63% decrease in MSE and at 30 minutes PWS 72% decrease in MAE from the Baselines, KF-CGA at 60 minutes and 30 minutes PWS achieves 40% and 72% decrease in MSE, MAE respectively. KF-CLA at PWS 15 minutes achieves a 72% decrease in MSE, and at PWS 30 minutes, an 77% decrease in MAE from the baselines. Figure 9 best results from Table 7 CPU utilization % of Planet lab dataset [36] in sub figures (d), (e), (f).

6.4 Microsoft Azure Dataset

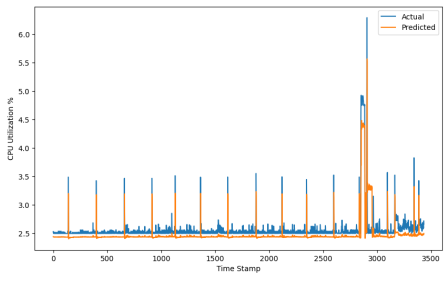
For our experiments we have considered Microsoft Azure 2019 dataset where we have considered few Virtual machine data out of 125. In this dataset average CPU utilization % is considered since the data provides minimum, maximum and average



(a) LSTM Cpu Util % PWS=1 second



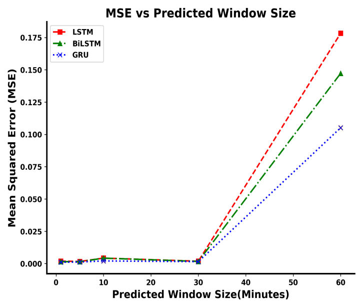
(b) Bi-LSTM Cpu Util % PWS=1 second



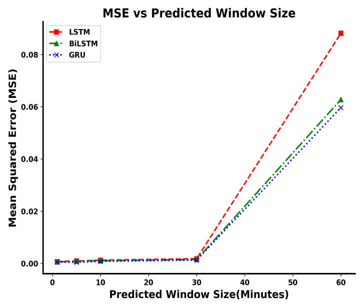
(c) GRU Cpu Util % PWS=1 second

Fig. 10: Best results from Table 9 Cpu Utilization% of Bitbrains Dataset

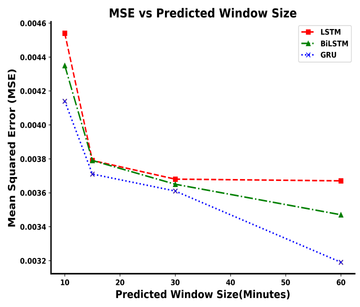
CPU utilization %. Using Microsoft Azure 2019 Trace data and a range of prediction window sizes (PWS), the Table 8 presents the mean percentage of CPU consumption (MSE and MAE) for various prediction models. Among KF-CLA and KF-CBiLA, KF-CGA continuously shows the lowest MSE and MAE values, demonstrating its greater prediction accuracy for average CPU utilization %. Furthermore, KF-CGA outperforms both KF-CLA and KF-CBiLA regarding prediction accuracy, with their performance being essentially equal. According to these results, the suggested models KF-CGA in particular offer viable substitutes for estimating average CPU utilization percentage in Microsoft Azure scenarios, possibly helping resource allocation and optimization efforts. KF-CLA at PWS 10 minutes achieves a 96% decrease in MSE, and at PWS 15 minutes, an 85% decrease in MAE . KF-CBiLA at 30 minutes PWS



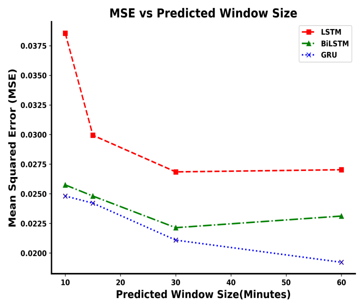
(a) Alibaba CPU_Util% MSE



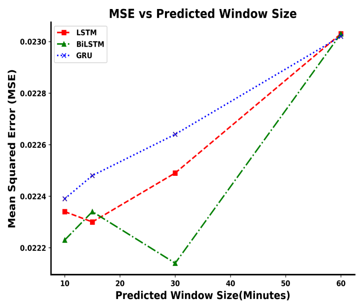
(b) Alibaba Mem_Util% MSE



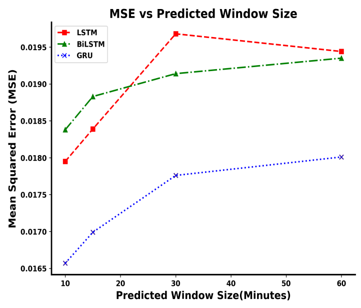
(c) Matrena CPU_Util% MSE



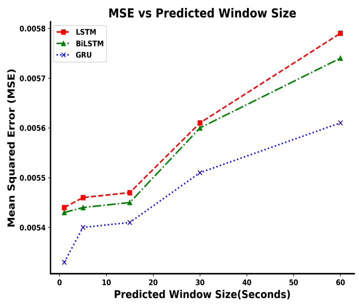
(d) Matrena Mem_Util% MSE



(e) Azure Average Cpu_Util% MSE



(f) Planet lab Cpu_Util% MSE



(g) Bitbrains Cpu_Util% MSE

Fig. 11: MSE of a) Alibaba Datasets CPU utilization % b) Alibaba Datasets Memory utilization% c) Matrena Datasets CPU utilization % d) Matrena Datasets Memory utilization% different time intervals

achieves 77% decrease in MSE and at 15 minutes PWS 83% decrease in MAE from the Baselines, KF-CGA at 10 minutes and 15 minutes PWS achieves 96% and 86% decrease in MSE, MAE respectively. Figure 9 best results from Table 8 Average CPU utilization % of Microsoft Azure dataset [37] in sub figures (a), (b), (c).

6.5 Bitbrains Dataset

With varying prediction window widths (PWS), Bitbrain's Trace data generates many prediction models, each yielding distinct MSE and MAE values for CPU use percentage. When compared to KF-CBiLA and KF-CGA, KF-CLA consistently shows the lowest MSE and MAE values, demonstrating its greater predictive accuracy for CPU utilization %. In particular, KF-CLA outperforms the other MSE and MAE value reduction models, particularly at larger PWS values. However, KF-CBiLA and KF-CGA also perform competitively, displaying comparatively low MAE and MSE values across several PWS. It is interesting that KF-ARIMA performs worse than the suggested models, particularly in percentage decrease, even if it produces lower MSE and MAE values for CPU(WK) and CPU(WoK). These results imply that the suggested models—KF-CLA in particular—offer viable substitutes for precisely forecasting CPU utilization% in Bitbrain's environment, potentially aiding resource management and optimization efforts. KF-CLA at PWS 1 second achieves a 91% decrease in MSE, and at PWS 5 seconds, an 75% decrease in MAE . KF-CBiLA at 1 minute PWS achieves 87% decrease in MSE and 84% decrease in MAE from the Baselines, KF-CGA at 5 seconds achieves 91% and 72% decrease in MSE, MAE respectively. Figure 10 with sub figures (a), (b), (c) shows best results from Table 9 CPU utilization % of Bitbrains dataset[38]

6.6 Performance

For Alibaba, KF-CGA demonstrated the best performance for CPU usage at PWS 1 minute and Memory at PWS 5 minutes. In contrast, KF-CLA and KF-CBiLA demonstrated comparable performance with ideal results at PWS of 5 minutes. Where as for Materna's trace KF-CGA demonstrated the highest performance at PWS 60 minutes, KF-CLA and KF-CBiLA yielded the lowest MSE and MAE. All models demonstrated good performance at PWS 60 minutes. For the Planet Lab trace At PWS 10 minutes, KF-CLA outperformed KF-CBiLA and KF-CGA, which had comparable performances. Regarding Azure Trace Across a range of PWS values, KF-CLA, KF-CBiLA, and KF-CGA obtained comparable MSE and MAE values. Regarding Bitbrains trace, All models demonstrated competitive performance across a range of PWS values, much like Azure. From the Table 6 we can conclude that KF-GRU performed better in identifying long-term dependence's in the Materna dataset.

6.7 PWS Accuracy

The best PWS for KF-CLA and KF-CBiLA in the Alibaba dataset was 5 minutes, but the best PWS for KF-CGA was 1 minute for CPU utilization and 5 minutes for memory consumption. The ideal PWS for every model in the Materena dataset was 60 minutes. While the optimal PWS varied in the Azure Trace dataset, with KF-CLA and KF-CBiLA exhibiting comparable performance across all PWS values, Planet Lab demonstrated an ideal PWS of 10 minutes for all models. On the other hand, for every model in the Bitbrains dataset, the ideal PWS was 1 second, since the data captured every 3 millisecond.

6.8 Tradeoffs

KF-CGA showed a possible trade-off between efficiency and PWS accuracy in the Alibaba dataset, with competitive performance at lower PWS levels. On the other hand, all models in the Materna dataset demonstrated a steady performance across a range of PWS values by striking a compromise between accuracy and efficiency. While other models produced competitive results, KF-CLA obtained the best balance and demonstrated optimal performance at PWS 10 minutes for the Planet Lab dataset. KF-CGA balanced accuracy and processing economy by achieving competitive performance in the Azure Trace dataset with a shorter PWS. Similarly, all models in the Bitbrains dataset showed a balance between efficiency and accuracy, with PWS 1 second showing the best results.

6.9 Robustness

KF-CGA had comparatively constant performance over a range of PWS values in the Alibaba dataset, while KF-CLA and KF-CBiLA showed a performance decline at higher PWS levels. In contrast, KF-CGA showed steady performance independent of PWS values in the Materena dataset, whereas KF-CLA and KF-CBiLA consistently improved with longer PWS. All models in the Planet Lab dataset showed a continuous improvement with longer PWS, demonstrating their resilience to variations in prediction window size. Similarly, all models demonstrated stable performance across a range of PWS values in both the Azure Trace and Bitbrains datasets, with negligible variations in MSE and MAE, highlighting their resilience.
The models usually performed competitively over a range of PWS values. Figure 11 gives 5 datasets MSE against the PWS. KF-CGA frequently demonstrated a possible trade-off between accuracy and efficiency, even if each dataset had unique subtleties. While KF-CGA showed consistent performance throughout a range of PWS values, KF-CLA and KF-CBiLA showed strong performance with longer PWS. Overall, KF-CLA and KF-CBiLA may be preferable for longer-term forecasts. At the same time, KF-CGA may be helpful for shorter-term forecasting due to its possible better computational efficiency, considering the trade-offs between accuracy efficiency and resilience.

7 Conclusion

Forecasting workloads at Edge Servers for cloud service providers, using a hybrid novel Kalman filter with CNN and attention-based LSTM, BI-LSTM, and GRU models, has been proposed. Extensive evaluation on real-world datasets like Microsoft Azure 2019, Bitbrains, Alibaba v2018, Materna, and Planet Lab has shown that these suggested models notably enhance the accuracy of workload prediction by significantly lowering the MSE. MSE is significantly decreased by the KF-CLA, KF-CBiLA, and KF-CGA models, which regularly outperform baseline methods. Reductions in MSE were noted for the following Cloud traces: Alibaba 97%, 82%, and 90%; 73%, 73% and 63% Materna; Planet Lab 72%, 63%, 40% Microsoft Azure 95%, 77%,96% and Bitbrains; 91%, 87%, and 91%. These results highlight the effectiveness of deep learning architectures for workload prediction in cloud and edge computing scenarios when combined with cutting-edge methods like attention mechanisms and Kalman filtering. The suggested models present a viable strategy to address the difficulties in managing resources in cloud and edge contexts. Cloud service providers can improve resource allocation, reduce waste of computing and storage resources, and guarantee adherence to users' QoS criteria by precisely anticipating workloads. The suggested models present a viable strategy to address the difficulties in managing resources in cloud and edge contexts. As future work, these traces will be evaluated in a federated learning framework to scale the applications in the inter-cloud scenario and avoid vendor lock-in.

List of Abbreviations

QoS	Quality of Service
KF-CLA	Kalman Filter based Convoluaional Long Short Term Memory with Attention
KF-BiLA	Kalman- Filter based Convoluaional Bi-directional Long Short Term Memory with Attention
KF-CGA	Kalman- Filter based Convoluaional Gated Recurrent Unit with Attention
EDC	Edge Data Center
ARIMA	Auto Regressive Integrated Moving Average
SVM	Support Vector Machines
LSTM	Long Short-Term Memory
Bi-LSTM	Bi directional Long Short-Term Memory
GRU	Gated Recurrent Unit
RNN	Recurrent Neural Network
CNN	Convoluational Neural Network
DNN	Deep Neural Network
DBN	Deep Belief Network
ANN	Artificial Neural Network
ML	Machine Learning
NN	Neural Network
IoT	Internet of Things
SaaS	Software as a Service
GCP	Google Cloud Platform
MSE	Mean Squared Error
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
AR	Auto Regressive
MA	Moving Average

BIC	Bayesian Information Criterion
AIC	Akaike Information Criterion
PWS	Predicted Window Size
ReLU	Rectified Linear Unit
WK	With Kalman
WoK	Without Kalman
SLA	Service Level Agreement
EDC	Edge Data Center

- **Author Contributions** The authors confirm contribution to the paper as follows: study conception and design: Naveen Kumar M R, Vishwas Yadav; data collection: Naveen Kumar M R, Vishwas Yadav; analysis and interpretation of results: Naveen Kumar M R, Vishwas Yadav; draft manuscript preparation: Naveen Kumar M R, Annappa B, Vishwas Yadav; Supervision: Annappa B. The final version of the manuscript approved by all authors.
- **Funding** The authors did not receive support from any organization for the submitted work.
- **Availability of data and materials** The authors confirm that the data supporting the findings of this study are available.
- **Ethical and informed consent for data used** The data used in this investigation from publicly accessible sources; therefore, obtaining informed permission and addressing ethical concerns were not necessary.
- **Conflict of Interest** The authors declare that they have no conflict of interest.

References

[1] Nguyen, C., Klein, C., Elmroth, E.: Multivariate lstm-based location-aware workload prediction for edge data centers. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 341–350 (2019). IEEE

[2] Kashyap, S., Singh, A.: Prediction-based scheduling techniques for cloud data center's workload: a systematic review. Cluster Computing **26**(5), 3209–3235 (2023)

[3] Li, Y., Yuan, H., Fu, Z., Ma, X., Xu, M., Wang, S.: Elastic: edge workload forecasting based on collaborative cloud-edge deep learning. In: Proceedings of the ACM Web Conference 2023, pp. 3056–3066 (2023)

[4] Calheiros, R.N., Masoumi, E., Ranjan, R., Buyya, R.: Workload prediction using arima model and its impact on cloud applications qos. IEEE Transactions on Cloud Computing **3**(4), 449–458 (2015) https://doi.org/10.1109/TCC.2014.2350475

[5] Saxena, D., Kumar, J., Singh, A.K., Schmid, S.: Performance analysis of machine learning centered workload prediction models for cloud. IEEE Transactions on

Parallel and Distributed Systems **34**(4), 1313–1330 (2023)

[6] Devi, K.L., Valli, S.: Time series-based workload prediction using the statistical hybrid model for the cloud environment. Computing **105**(2), 353–374 (2023)

[7] Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems **28**(1), 155–162 (2012)

[8] Ho, S.-L., Xie, M., Goh, T.N.: A comparative study of neural network and box-jenkins arima modeling in time series prediction. Computers & Industrial Engineering **42**(2-4), 371–375 (2002)

[9] Huang, H.-C., Cressie, N.: Spatio-temporal prediction of snow water equivalent using the kalman filter. Computational Statistics & Data Analysis **22**(2), 159–175 (1996)

[10] Chen, J., Wang, Y., et al.: A hybrid method for short-term host utilization prediction in cloud computing. Journal of Electrical and Computer Engineering **2019** (2019)

[11] Yao, F., Yao, Y., Xing, L., Chen, H., Lin, Z., Li, T.: An intelligent scheduling algorithm for complex manufacturing system simulation with frequent synchronizations in a cloud environment. Memetic Computing **11**(4), 357–370 (2019)

[12] Kumar, J., Singh, A.K.: Performance evaluation of metaheuristics algorithms for workload prediction in cloud environment. Applied Soft Computing **113**, 107895 (2021) https://doi.org/10.1016/j.asoc.2021.107895

[13] 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 (2022)

[14] Xiao, D., Cao, B., Wu, W.: Efl-wp: Federated learning-based workload prediction in inter-cloud environments. In: 2022 International Joint Conference on Neural Networks (IJCNN), pp. 1–10 (2022). https://doi.org/10.1109/IJCNN55064.2022.9892264

[15] Tran, M.-N., Vu, X.T., Kim, Y.: Proactive stateful fault-tolerant system for kubernetes containerized services. IEEE Access **10**, 102181–102194 (2022) https://doi.org/10.1109/ACCESS.2022.3209257

[16] Yadav, M.P., Pal, N., Yadav, D.K.: Workload prediction over cloud server using time series data. Proceedings of the Confluence 2021: 11th International Conference on Cloud Computing, Data Science and Engineering, 267–272 (2021) https://doi.org/10.1109/Confluence51648.2021.9377032

[17] Khan, S.A., Abdullah, M., Iqbal, W., Butt, M.A., Bukhari, F., Hassan, S.-U.: Automatic migration-enabled dynamic resource management for containerized workload. IEEE Systems Journal **17**(2), 2378–2389 (2023) https://doi.org/10.1109/JSYST.2022.3204748

[18] Bi, J., Li, S., Yuan, H., Zhou, M.: Integrated deep learning method for workload and resource prediction in cloud systems. Neurocomputing **424**, 35–48 (2021)

[19] Bai, Y., Chen, L., Lei, Y., Xie, H.: A deep learning prediction approach for machine workload in cloud computing. In: 2023 5th International Conference on Data-driven Optimization of Complex Systems (DOCS), pp. 1–8 (2023). IEEE

[20] 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 Journal on Wireless Communications and Networking **2019**, 1–18 (2019)

[21] Ouhame, S., Hadi, Y., Ullah, A.: An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model. Neural Computing and Applications **33**(16), 10043–10055 (2021)

[22] Song, B., Yu, Y., Zhou, Y., Wang, Z., Du, S.: Host load prediction with long short-term memory in cloud computing. The Journal of Supercomputing **74**, 6554–6568 (2018)

[23] Dogani, J., Khunjush, F., Mahmoudi, M.R., Seydali, M.: Multivariate workload and resource prediction in cloud computing using cnn and gru by attention mechanism. The Journal of Supercomputing **79**(3), 3437–3470 (2023)

[24] Maiyza, A.I., Korany, N.O., Banawan, K., Hassan, H.A., Sheta, W.M.: Vtgan: hybrid generative adversarial networks for cloud workload prediction. Journal of Cloud Computing **12**(1), 97 (2023)

[25] Simaiya, S., Lilhore, U.K., Sharma, Y.K., Rao, K.B., Maheswara Rao, V., Baliyan, A., Bijalwan, A., Alroobaea, R.: A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. Scientific Reports **14**(1), 1337 (2024)

[26] Tran, M.-N., Vu, X.T., Kim, Y.: Proactive stateful fault-tolerant system for kubernetes containerized services. IEEE Access **10**, 102181–102194 (2022)

[27] 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 Transactions on Internet Technology (TOIT) **22**(3), 1–24 (2022)

[28] Al-Asaly, M.S., Bencherif, M.A., Alsanad, A., Hassan, M.M.: A deep learning-based resource usage prediction model for resource provisioning in an

autonomic cloud computing environment. Neural Computing and Applications **34**(13), 10211–10228 (2022)

[29] Xiao, D., Cao, B., Wu, W.: Efl-wp: Federated learning-based workload prediction in inter-cloud environments. In: 2022 International Joint Conference on Neural Networks (IJCNN), pp. 1–10 (2022). IEEE

[30] Chen, L., Zhang, W., Ye, H.: Accurate workload prediction for edge data centers: Savitzky-golay filter, cnn and bilstm with attention mechanism. Applied Intelligence **52**(11), 13027–13042 (2022)

[31] Yadav, V.: workload prediction . https://github.com/Vishwas-yadav/workload_prediction (2024)

[32] Sun, W., Xu, X.: Aledar: An attentions-based encoder-decoder and autoregressive model for workload forecasting of cloud data center. In: 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 59–64 (2022). https://doi.org/10.1109/CSCWD54268.2022.9776279

[33] Shukla, A., Kumar, S., Singh, H.: Mlp-ann-based execution time prediction model and assessment of input parameters through structural modeling. Proceedings of the National Academy of Sciences, India Section A: Physical Sciences **91**(3), 577–585 (2021) https://doi.org/10.1007/s40010-020-00695-9

[34] Alibaba(2018). https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018. Accessed: April 20, 2024 (2024)

[35] Materna. http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna. Accessed: April 20, 2024 (2024)

[36] PlanetsLab. https://github.com/beloglazov/planetlab-workload-traces. Accessed: April 20, 2024 (2024)

[37] MicrosoftAzure. https://github.com/Azure/AzurePublicDataset. Accessed: April 20, 2024 (2024)

[38] Bitbrains. http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains. Accessed: April 20, 2024 (2024)