Precision Offloading in Edge Computing: Leveraging Predictive Model

[1]Suganya T S, [2]Saivijayalakshmi J, [3]Karthik M,[4]Keerthana T, [5]Srikanth V and [6]Vidhya U
[1,2]Department of Computer Science and Applications, SRM Institute of Science and Technology, Ramapuram Campus, Chennai, Tamil Nadu, India.
[3]Department of Artificial Intelligence and Data Science, K.Ramakrishnan College of Engineering, Samayapuram, Trichy, Tamil Nadu, India.
[4]Department of Computer Science and Engineering, Chennai Institute of Technology, Chennai, Tamil Nadu, India.
[5]School of Science and Computer Studies, CMR University OMBR Layout, Satellite Campus, Bengaluru, Karnataka India.
[6]Department of Computer Science and Engineering, OASYS Institute of Technology, Trichy, Tamil Nadu, India.
[1]tssuganya07@gmail.com, [2]vidhyasuresh74@gmail.com , [3]karthikm.careers@gmail.com, [4]keerthana64t@gmail.com, [5]srikanth.v@cmr.edu.in, [6]ammuvidhyait@gmail.com.

Correspondence should be addressed to Karthik M : karthikm.careers@gmail.com.

Article Info
Journal of Machine and Computing (http://anapub.co.ke/journals/jmc/jmc.html)
Doi : https://doi.org/10.53759/7669/jmc202404100
Received 18 May 2024; Revised from 30 July 2024; Accepted 06 August 2024
Available online 05 October 2024

Abstract – The intended effect of the investigation is to provide sophisticated prediction and decision-making models in order to optimize service delivery and improve the Quality of Experience (QoE) for users. This research tackles the problems that are associated with job offloading in edge computing settings. In order to reduce service latency and improve overall performance, the Bi-Directional Long Short-Term Memory (B-LSTM) model is used. This model provides the ability to forecast task creation and server load. In order to accommodate the particular qualities of different devices, the Selective Objective Offloading Decision (SOOD) approach is presented. This method makes use of the TOPSIS methodology to turn server assessment into a decision-making issue that involves several criteria. A considerable increase of 98.4% in user quality of experience is achieved by the SOOD paradigm. In addition, the Rapid Offloading Decision (ROD) model is presented in order to manage unexpected work patterns. This is accomplished by using the log information of surrounding devices, which results in instantaneous and dependable offloading choices. Through the usage of prediction algorithms and selective decision-making, this research gives a complete strategy to improving the efficiency of edge computing. The goal of this technique is to maximize the utilization of servers and the user experience.

Keywords – IoT, Deep Learning, B-LSTM, QoE, SOOD.

I. INTRODUCTION

Computational nodes are devices that process the data that they generate or acquire from the environment. These devices can be things like cell phones, laptops, desktops, Internet of Things (IoT) devices, sensor devices, etc. Due to recent developments in the configuration of these devices, the processing of the data mostly happens with the devices. But data analysis demands higher levels of processing, like fast Cloud computing has transformed contemporary businesses by offering crucial infrastructure and processing services, allowing devices to assign jobs to the cloud for processing and storage. In order to improve the availability and performance of services, edge computing deploys servers in close proximity to customers, resulting in decreased latency, lower energy use, and similar advantages to those provided by cloud computing. Nevertheless, in order to maximize server efficiency and enhance the Quality of Experience (QoE) for users, the process of job offloading to the edge must be carefully and strategically organized. Effective task prediction on the device and load prediction on the server are necessary for this. Conventional methods often result in longer service delays because of the time required to evaluate server status when a job is initiated. In order to tackle this issue, the Bi-Directional Long Short-Term Memory (B-LSTM) model is used to forecast task creation and server load. Although generic offloading solutions provide a comprehensive solution, they fail to consider the unique characteristics of individual devices. In order to address this constraint, the suggested approach is the Selective Objective Offloading Decision (SOOD) technique, which use the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) to assess server factors and convert them into a multi-criteria decision-making issue. The SOOD model has exceptional performance, attaining a 98.4% enhancement in user

Quality of Experience (QoE) in comparison to other models. Cloud settings are still faced with challenges due to irregular work patterns in specific devices, notwithstanding the gains made. In order to address this issue, the Rapid Offloading Decision (ROD) model is used, which leverages log data from nearby devices to provide prompt and dependable resolutions for devices that deviate from the norm. The ROD model efficiently manages unforeseen and limited occurrences, hence improving the precision of the offloading process. This study introduces a thorough strategy for transferring workloads to edge settings. It combines prediction algorithms and ranking approaches to enhance service delivery efficiency while ensuring a high quality of experience. response, heavy storage, and big data analysis. However, devices fail to achieve the desired output by the deadline due to their limitations in computation and storage. Cloud computing emerges as a solution to combat these challenges. Cloud technology will provide users with computer resources on client specific needs and on basis of price per service. The cloud provides services like storage, processing servers, networking, intelligence, and data analytics [1]. This facilitates the rapid development of the organisation by providing an instant platform according to the evolution of the application[2][3]. The overhead associated with the management of resources is transferred from the users to the cloud service providers.

SaaS

A method of providing applications and software's over the Internet as a service is known as "Software as a Service" (or SaaS). Hosted service and on-demand software are some of the names for SaaS; all state that accessing software through a browser without owning it. The users will be able to utilise the software applications remotely through the internet facility. The client can benefit a lot with less investment and also avoid complicated software and hardware management by simply using software over the Internet instead of developing and monitoring the product. SaaS services are managed by a third-party vendor and delivered to their users. Examples of SaaS are email, Google Workspace (formerly GSuite), Dropbox, and Salesforce. GoToMeeting, etc.

PaaS

PaaS (Platform as a Service) uses a similar delivery model to SaaS, with the exception that it offers a platform for developing software rather than delivering software over the internet. Since this platform is delivered over the internet, developers are free to focus on creating the software rather than having to worry about infrastructure, storage, software updates, or operating systems. PaaS clouds are frequently used by experts in the field of coding to quickly create, modify, and test new computing programs. Example: Windows Azure, Heroku, Google App Engine, OpenShift Users who use PaaS can manage and develop applications on a shared cloud platform without having to create and maintain the infrastructure typically needed for the process. **Fig 1** shows Types of Cloud Computing.

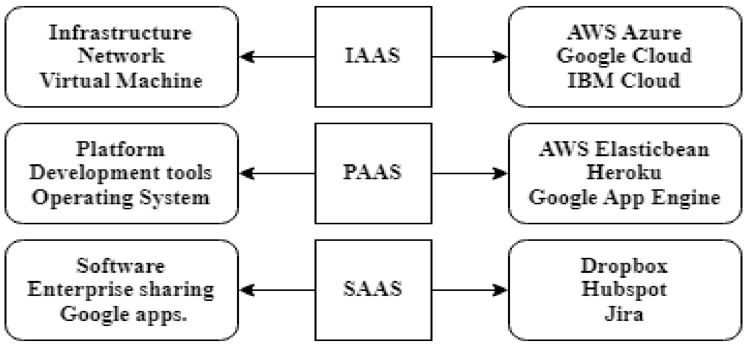

Fig 1. Types of Cloud Computing.

IaaS

In IaaS (Infrastructure as a Service), the hardware related to information technology is provided to the users so that it can be accessed remotely. A flexible service among all in cloud service is IaaS. Services for construction of the organization and productions of products are provided for lease by IaaS cloud providers. These IaaS components and services are available online for businesses. The user essentially rents the infrastructure and has access via an API (Application Programming Interface) or dashboard. IaaS is used for accessing and managing computer, networking, storage, and other services. Instead of having to purchase hardware outright, IaaS enables businesses to buy resources as needed and on demand. Examples are Rackspace, Cisco Metacloud, and Google Compute Engine (GCE).

XaaS

The abbreviation "XaaS," or "Everything as a Service," is one phrase you'll probably hear more often in the world. XaaS refers to adaptable, customised apps and services that are entirely controlled by users and the data they supply through widespread input gathering devices like smart devices and IoTs. By utilising the data created over the cloud, businesses may develop more quickly, improve their customer interactions, and extend the limit of the customer purchase. XaaS is a crucial resource for the digitalized robotic enterprise.

The QoE of the device in the network is the major objective of developing the model. Each and every device has unique needs, and satisfying them is the ultimate goal. The overall performance of the network is increased by proper offloading decisions for individual devices. The energy of the device and load balance of the server have also been optimised by utilising machine learning and genetic algorithms. Selective offloading of decision-making for computational nodes yields a better solution than decision for the entire network.

The task type is predicted for the device in advance; this will help prepare the server in advance and also make a decision on where to offload. The prediction is made based on the past history of the device and by analysing its request type and time. B-LSTM (Bi-Directional Long Short-Term Memory) is a learning algorithm that predicts the future upon analysis of past input. It takes input in both directions (forward and backward) and provides accurate predicted output. Load prediction on the server side has also been done to provide data for decision-making.

Supervised Learning

Supervised learning is learning the relationship between input data and output patterns by predicting the output based on the previous learned pattern. Training data are used to find the pattern and output for the new input data [4]. Support Vector Machine (SVM), Apriori, Neural Networks, Linear Regression, Particle Swarm Optimization (PSO), etc. are some of the well-known supervised algorithms.

Unsupervised Learning

Unsupervised learning is a model of learning without any initial training dataset. It usually reduces the dimension of the dataset by grouping them into labels based on the similarities, attributes, and differences [5]. Unsupervised algorithms are more complex than supervised algorithms, but they perform better in situations where prior knowledge is limited. K-means, Analytic Hierarchy Process (AHP), Hidden Markov Model (HMM), and clustering are some of the well-known unsupervised algorithms.

Reinforcement Learning

Reinforcement learning is a learning method based on the trial-and-error technique that consists of an agent and an environment. Agents act as learners and as decision-makers based on inferences from the environment [6]. Decision making functions in agents have policy functions that reward and punish based on the action of the previous state, which in turn supplies feedback for the next state.

Prediction and Analysis Methods

Cloud computing technology uses machine learning algorithms to solve a variety of issues. It offers suggestions for enhancing the cloud network's overall operation by analyzing the network's operation and forecasting numerous future events [7]. Offloading techniques involve choosing a server to handle the task; the server could be a device, an edge server, or a cloud server. ML algorithms support decision-making by examining network workflow and feedback from devices and servers. To identify the in-depth analysis of the task processing and modify the server selection based on feedback, ML algorithms like Deep Reinforcement Learning (DRL) and Convolutional Neural Networks (CNN) are implemented. An analysis model based on machine learning is implemented to eliminate security related issues.

Predictive model is also been achieved by using machine learning algorithms. By taking consideration on historical and recent information on system, predictive modelling uses machine learning and data interpretation to predict and give Intel on events occurs in future [8]. Decisions regarding offloading are based on a variety of variables, the values of which change over time. Analyzing changes and predicting future values will provide data for the algorithms that make decisions. Some of the problems that are resolved by using prediction models include task generation, load prediction, and path finding. In cloud computing, prediction algorithms like decision trees, K-means, random forests, and time series models are frequently used.

Individual devices are noted for their task requests, and the offloading decision is made by considering the server status by agent. Deep reinforcement learning on the status of the server and the experience of the device is monitored and stored. A penalty and reward are given based on the experience of the device and their request. A constant update of this status is forwarded to the agent for use in the decision-making algorithm. The ranking of the server for the device request by applying the TOPSIS algorithm picks the appropriate server. The ranking of servers is done by taking the priorities of the device request and then obtaining the values for servers from the agent.

Rapid offloading decisions are applicable in certain scenarios where a new device joins the network and needs offloading decisions without conducting any analysis. This has also been used by devices with energy constraints, which need a fast result for offloading without compromising the result. A rapid decision is made with the help of a simple request from the neighbouring devices and cross verifying the received response. Network optimization is achieved, and the quality of the experience on the device is also increased by the proposed model.

II. RELATED WORK

Cloud computing is the on-demand servicing of customers throughout the world in different categories through the internet. Offloading tasks from the devices to the cloud for the needs of storage, processing, security, monitoring, controlling, and many other services. Cloud server performance is improved by adding two new ideas Edge and Fog computing. Fog servers and Edge servers contribute to the fast connectivity of devices to the cloud by reducing the latency in communication and bringing almost every resource closer to the users. The selection of the server determines the quality of the experience on the devices. Many factors determine the selection process in a network, and many methods have also been developed.

Zhao et al. [9] used an Auto Regressive Integrated Moving Average Back Propagation Selective Offloading model (ABSO) is used to find the computation capability of the edge nodes and select the best node for offloading. Selective offloading is done by this model, which uses linear prediction and non-linear prediction using Autoregressive Integrated Moving Average (ARIMA) and Back Propagation (BP) algorithms. The prediction of load and resource capacity of the servers is given by this model, which provides an accurate value for selecting the best server. A back propagation based neural network is developed to identify the fault and advance to the next round of prediction.

For a multi-site heterogeneous cloud environment, Goudarzi et al. [10] implied a genetic algorithm to find by offloading to save energy and increase completion time. An initial population is created based on available servers that have limitations in completing the task in a timely manner, and then, based on training and prediction, their weightage is updated to analyse their limitations in solving the next task provided to them. The proposed Genetic Algorithm (GA) permits some random jumps when searching for the global optimum to avoid the local optimum problem and reduce the convergence time towards a global optimum. Reinforcement learning helps in each step of the algorithm to provide progressions for achieving the global optimum.

Rodriguez et al. [11] applied the PSO algorithm to their consideration of obtaining a timely response for deadline-based tasks. They modified the PSO algorithm by adding parallel processing of virtual machines for each different task in a request. To address the heterogeneity problem, they utilised multiple Virtual Machines (VM) to perform different tasks in parallel.

A similar solution is given by Huynh et al. [12] to solve the offloading problem in mobile edge computing by splitting the offloading problem into two: one computation resource allocation and a second computation offloading decision. Joint resource allocation and offloading models are used in multi-user, multi-server mobile edge computing by utilising the meta-heuristic PSO algorithm. The first problem of computation resource allocation is solved using Karush–Kuhn–Tucker conditions, which will solve the energy consumption problem of devices. Binary PSO is applied for solving the computation offloading decision problem, which decides whether to offload to a remote server or solve within the device using available resources.

The hybrid quantum behaved particle swarm technique was implemented by Dai S et al. [13] with two requirements: first, the total subcarrier is restricted; and second, the completion time must be smaller than the time required for local computing. The mixed integer nonlinear programming model with the shortest completion time is formulated. As the objective function is nonlinear and the constraints have integer terms, the problem is extremely complex. Two conditions are linked as the second condition is solved by the water filling algorithm, which in turn helps to solve the first condition. A hybrid quantum based PSO algorithm is developed to solve the objective problem and provide the shortest time to completion.

In the work by Darbanian et al. [14], the best Fog Device is selected for the module placement based on the features and parameters of fog devices internal configuration. Authentication, confidentiality, integrity, availability, capacity, speed, and cost are the criteria that were used for classification of fog devices. Classifiers like Decision Tree, Random Forest, Extra-trees, and AdaBoost are implemented to categorise the devices. Overall, better response time is increased by the classification of Fog Devices.

Partial offloading for Augmented Reality (AR) tasks is done by using heuristic algorithms, and an integer-based PSO algorithm is developed by Liu et al. [15]. Partitioning the heavy tasks into subtasks and offloading them partially to the edge side and other less computational tasks to the user device itself a heuristic graph-based algorithm is used to cluster the tasks and send them to the edge; this will reduce the delay on multiple tasks. Improved PSO algorithm is used to find optimal solution by taking CPU allocated to the task and their distance and velocity of the particle.

Similarly, Hui Qi et al. [16] applied three learning algorithms at different levels to achieve the overall performance of the multi-tier cloud architecture. First, k means algorithm is used to cluster the physical machines based on their availability in close range; second, the best cluster for a particular task is selected by using deep reinforcement learning, which uses the metric from the first algorithm; and the third step involves finding the optimal machine to serve the offloaded task is find by using the improved PSO algorithm. This model applies learning machine in cloud and also in edge to perform the proposed operation.

III. METHODOLOGY

Offloading Prediction Model

A prediction model's objective is to find patterns in previous data and use those patterns to predict future occurrences with accuracy. Simple linear regression models and more sophisticated machine learning models like neural networks and decision trees can all fall into this category. Predicting the tasks or workloads that should be transferred from a device or system to a remote resource, such as a cloud server or edge computing device, is known as offloading prediction. Offloading is a popular method in distributed computing that boosts efficiency and uses fewer resources. A number of variables, including the present workload on the device, the processing power of the remote resource, the state of the network, and the priorities of the device, must be taken into account in order to create accurate offloading forecasts. To construct offloading prediction models that analyse these variables and foretell which tasks should be delegated for optimum performance and resource utilisation, machine learning methods can be applied. These models can be improved over time to increase accuracy. They can be trained on historical data or in simulated situations.

Recurrent Neural Networks

A class of neural networks termed Recurrent Neural Networks (RNNs) is useful for modeling sequence data. RNNs can process input sequences using their internal state (memory), in contrast to feedforward neural networks. RNNs are recurrent in nature since they act on the similarly for all the information taken, and the outcome of the current input depends on the outcome of the prior computation. Prior to being sent to the following unit, the output is processed. By combining both inputs, the outcome is ultimately selected. The output from the neuron is decided by the input data and the output from the previous neuron, as this procedure helps in remembering the context of the information pattern. Prediction is achieved by using the pattern and upcoming input of the system.

Fig 2 Illustrates how the RNN process the input of each state and predict the output using the previous state analysis. Application of RNN includes speech recognition, forecasting, and virtual assistant's software's. weight of the input and hidden state are also been used to tune the prediction process. Tanh function is used for modeling the activation unit, which uses -1 to +1 as range for values.

Any unfolding RNN is trained across a number of time steps, and the error gradient is determined as the total of all gradient errors over timestamps. Since the chain rule is used to calculate the error gradients, the multiplicative term dominates over time, which has an explosive or disappearing effect on the gradient. Gradient clipping can be used to stop gradients from bursting. The gradients are clipped, as the name implies, if they cross a predetermined threshold. However, the issue of vanishing gradients still exists. Using relu and tanh makes this model slow and hard to process the inputs with long sequence.

Long Short-Term Memory

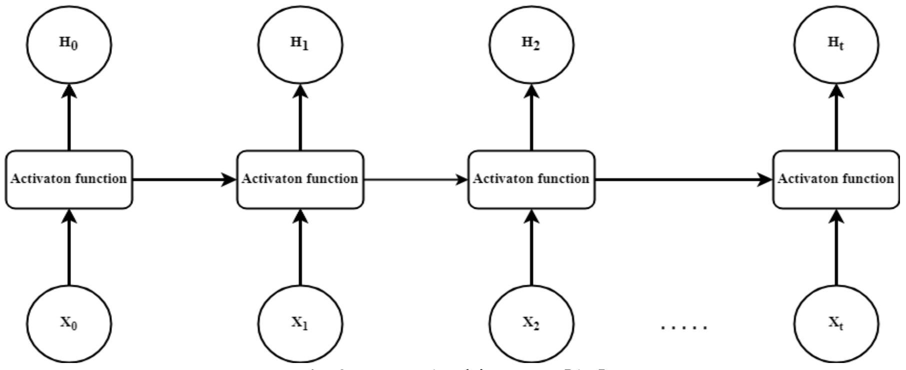

Fig 2. RNN Architecture [17].

Recurrent neural networks (RNN) can process time sequence data because they can simulate the connections between the initial and final states [17]. Unfortunately, the "long dependency" problem in the basic RNN architecture prevents the RNN from processing a lengthy sequence. LSTM network, which can learn long-term dependencies, thereby offers better prediction. In addition to the standard RNN cell, the LSTM cell's architecture includes three gates, a hidden state, and a hidden state, as shown in **Fig 3.**

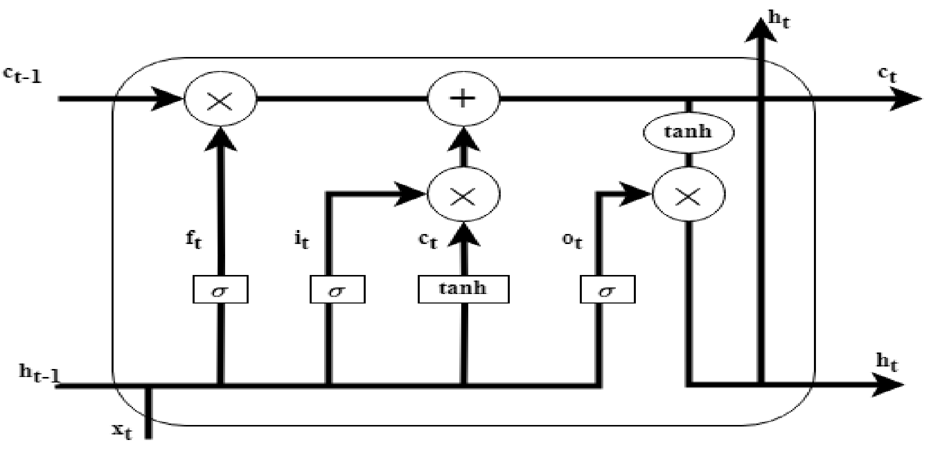

Fig 3. LSTM Architecture [17].

The crucial characteristic is that those straightforward LSTM networks can store data that can be used for next cell operations. Long-term state is used by LSTM, which records, reads, and discards items meant for long-term storage as they traverse the network. The output produced as a result of current execution is preserved by the short-term state. Accessing, storing, and rewriting decisions are made based on an activation function. Arriving new pieces of data are either kept or discarded depending on the output and forget gates' decisions. The memory of the LSTM block and the state of the output gate produces the model decision. The output is then used as an input by the network once more to create a recurrent sequence. Four neural networks, often known as cells, and different memory building elements make up the chain structure of the LSTM. Cells and gates both play a role in memory modification and information retention. Three gates are present:

Forget Gate

The purpose of forget gate is to clear unwanted information in the cell state. The Weights of the inputs (present cell-xt and previous cell -ht-1), are multiplied for calculating the output on that stage. The result decides whether to store or clear the information from the cell state. Value 1 indicates related information, that can be processed further, and value 0 indicates unwanted information, that has to be cleared.

Input Gate

Initial input and information from the previous cell state are processed by the input gate. The sigmoid function is used to control the inputs ht-1 and xt, to filter the information as in the forget gate process. The tanh function, is implemented to create vector form from the two inputs. Finally, the product of vector and output of gate yield valuable information for the next gate or output gate.

Output Gate

The output gate processes the input from previous gates and produces output according to it. Using the inputs ht-1 and xt, the information is then filtered by the values to be remembered. Activation functions are applied to clarify the irregularities in inputs. Output gates also provide input for the next state by processing inputs and regulating the functions for obtaining the desired output.

LSTM is implemented for workload prediction. Initially, each server in the cluster has its collected workload traces. The workload prediction model examines these traces and forecasts changes in workload throughout the ensuing period. The load balance server is then updated with a new allocation schedule. An LSTM-based encoder-decoder network and an output layer make up the model's two halves. In order to encode the time sequence data into the context vector, the encoder must first receive the time sequence data. The intermediate prediction results for the output layer are then produced iteratively by the decoder. The output layer then outputs the workload prediction values.

B-LSTM Prediction Model

B-LSTM is prediction model used to find patterns in sequential data by taking two hidden layers for increasing the accuracy. It considers both the past and future data for the prediction of future data. Forward hidden layer H_t^f and backward hidden layers H_t^b takes the input in forward and backward order respectively and the output of this layer is combined and send to next layer.

Forward layer is expressed as

$$H_t^f = \tanh\left(W_{xh}^f x_t + W_{hh}^f H_{t-1}^f + b_h^f\right) \tag{1}$$

Here H_{t-1}^f is the forward hidden state input, W is the weight matrix (W_{xh}^f is the weight merging input (x) to hidden layer (H)) in forward direction, b_h^f is a forward bias vector.

Backward layer is expressed as

$$H_t^b = \tanh\left(W_{xh}^b x_t + W_{hh}^b H_{t+1}^b + b_h^b\right) \tag{2}$$

Where H_{t+1}^b is the backward hidden state input, W is the weight matrix (W_{xh}^b is a weight merging input (x) to hidden layer (H)) in backward direction, b_h^b is a backward bias vector.

Output is calculated as

$$y_t = W_{hy}^f h_t^f + W_{hy}^b h_t^b + b_y^2 \tag{3}$$

Where W_{hy}^f gives the output of forward weighted matrix and W_{hy}^b gives the output of backward weighted matrix. Learning rate is set as 0.01 as at 0.0001 the convergence rate is less and takes longer time to obtain optimal, at 0.1 the fluctuation of loss is high which fall to false output. **Fig 4** shows Working of B-LSTM.

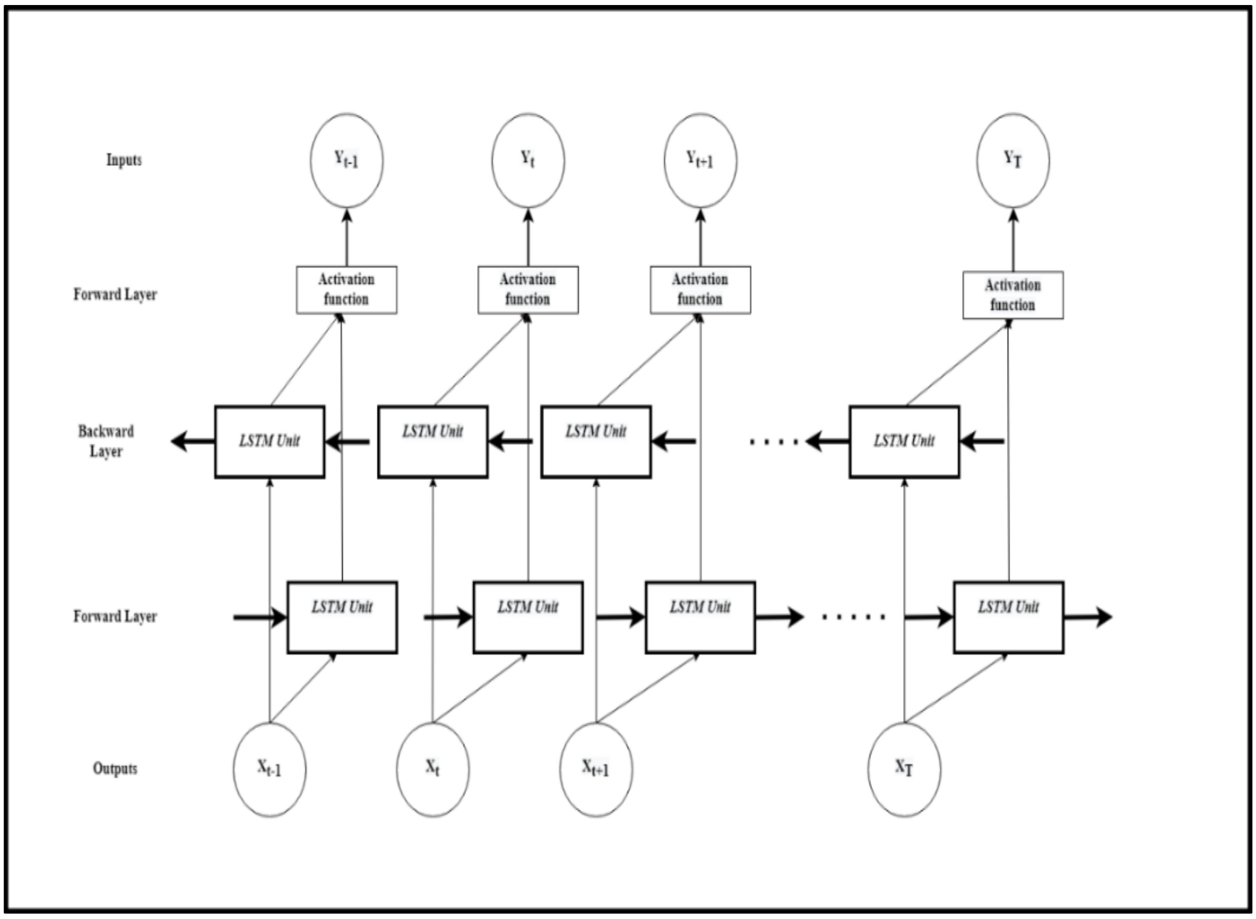

Fig 4. Working of B-LSTM [101].

Device Task Prediction

A decision-making process is required in computational offloading systems following task generation, and there will be some lag time between task generation and the decision-making process. Despite being a dynamic and random process, task generation will have a strong relationship with time because of its long-term nature. As a result, the tasks are predicted for user device network time slot based on user device history, and service packages are loaded in advance of the actual tasks arriving (e.g., allocate the best computation server in advance through the decision model). From the devices task information T_1, T_2, · · · T_n, the model is trained to predict more accurate value of the next task T_{n+1}. Hence, the model is effective in prediction only if $| T_{n+1} - T`_{n+1} |=0$, i.e., predicted and actual values are more or less equal.

Devices are initially set with priorities based on their requirements for edge service. Apart from general constraints like energy, delay, cost, etc., devices can set their operation-oriented priorities. For instance, healthcare device requests for storage are set with accuracy, reliability, and delay. A request for processing a scan report needs image processing; in that case, priority is set with accuracy, service type, and service time. [18] All devices are set priority for their request in $r_i \in fac(d_i)$. where $fac(d_i)$. contains all possible predefined request types. Tasks produced by the devices according to certain parameters satisfy the SLA requirements. Different devices have various parameters, each with a different priority. **Table 1** displays many task-specific parameters, together with information on the parameters' priorities. This demonstrates that tasks are only satisfied when their unique parameter list is met.

Table 1. Request Type of Different Application

Type	Priority	Applications
t_1=Type of service t_2=delay t_3=cost per CPU t_4=cost per storage t_5=failure rate t_6=accuracy t_7=security t_8=load	(t_4,t_1,t_2)	Storage
	(t_4,t_2,t_6)	Medical record access
	(t_2,t_5,t_1)	Banking, critical application
	(t_8,t_2,t_5)	Continuous access application
	(t_7,t_6,t_5)	Privacy application
	(t_2,t_8,t_2)	General applications

Tasks are characterised in many ways; task identification is the first step in a prediction model. A task is allocated to the computational or storage server based on its nature. Tasks that are needed to store the data are simply satisfied by a server with a lower cost per storage, a high access rate, and security. Tasks that concern processing the data and real-time interaction are of different types. Identifying the type of task and setting the parameter for prediction will increase the accuracy of the offloading process. **Fig 5** shows the process of task allocation based on the prediction of future tasks by the model. Upon the arrival of the real task, the error between the real task generated and the predicted task is compared with the threshold value. If the error is within the limit, then the task is offloaded as per the decision made by the agent; otherwise, a decision is made for the task using the ranking algorithm. The limit and the data from the new job are added to the model's training set.

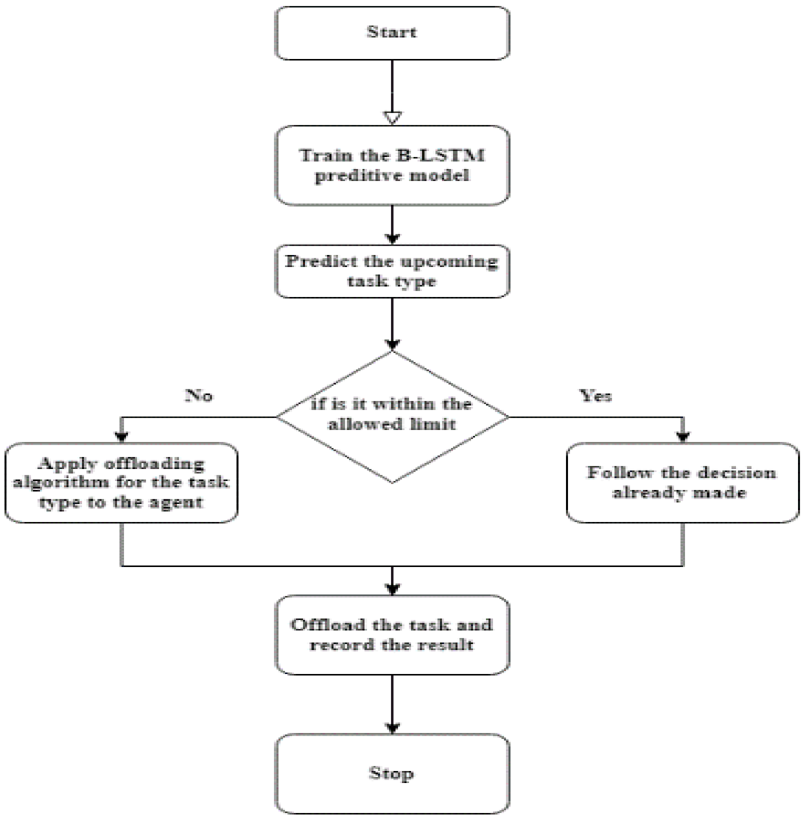
Start
Train the B-LSTM preditive model
Predict the upcoming task type
if is it within the allowed limit
No
Yes
Apply offloading algorithm for the task type to the agent
Follow the decision already made
Offload the task and record the result
Stop

Fig 5. Task Prediction Model.

Task Type Prediction Algorithm

Input: tt_{pa}^{w} past task request of size w
Output: tt_{pre}predicted task request type

```
1.   begin
2.   for each $tt_{pa}^{w}$in the sequence
3.        prepare the input set ($tt_{pa}^{w}, s$)
4.        $h_t^f$ and $h_t^b$are prepared
5.        Run prediction using B-LSTM
6.        Predict the outcome $tt_{pre}$
7.   If  $tt_{pre}$ - $tt_{act}$ < threshold
8.   offload according to cache
9.   update the Agent cache
10.  update the cache of device
11.  If  $tt_{pre}$ - $tt_{act}$ > threshold
12.       perform ROD
13.  update cache of agent and device
```

Serve Load Prediction

The heterogeneous nature of edge servers allows them to offer service at many levels. Because not all servers will be able to meet a device's needs similarly, offloading decisions are made to select the best one. The cloud agent continuously evaluates the performance of the servers in response to each device request to keep track of their status. Deep reinforcement learning learns the state of the servers including load prediction, latency, type of service, cost, accuracy, waiting time, success rate, etc. This server status E={E_1, E_2, E_3, . . . E_n} provides the status for ranking process in SOOD algorithm . The cache of the agent contains the status and applied for Multi Criteria Decision Making (MCDM). Based on the results future prediction for the servers are made.

Server Load Prediction Algorithm

```
Input: $sl_{pa}^{w}$ past load status of size w and $tt_{pre}$ predicted task
Output: $sl_{pre}$predicted server load
1.  begin
2.  for each $tt_{pre}$ in the range
3.  calculate the predicted selection by $n$ tasks $C_s^{rate}$
4.  $l_t = \sum_{r=0}^{n} C_s^{rate}$
5.  prepare the input set ($l_t$, $sl_{pa}^{w}$)
6.      $h_t^f$ and $h_t^b$are prepared
7.      Run prediction using B-LSTM
8.      Predict the outcome $sl_{pre}$
7.  update the Agent server status
9.  update the cache of device
```

If task prediction is not taken into account, poor offloading decisions may result in a load imbalance. The size of the task that has the probability of choosing the server for offloading will increase the prediction of load next time. The edge server's monitoring system logs system information and performance metrics, including CPU usage, network usage, and the tasks that are currently being executed. The logged information can be used to forecast future server load. The historical data is used to predict the edge server load level (the number of standby queues on the edge nodes) using the load model. The predicted idle server may be chosen in preference as the offload computing node. The efficiency of the server is determined by its processing speed, queue capacity, and task rate. Tasks are assigned to servers by the scheduler depending on the specifications of each server, which vary. By effectively scheduling the jobs, server load prediction lowers the offloading error rate. B-LSTM uses the server parameters to forecast the state of the forthcoming server load, which helps to balance job distribution and keep the server load balanced.

IV. RESULT ANALYSIS

The Simulation is set up by using the Edgecloudsim package in Eclipse 2021. Edgecloudsim is an edge cloud network simulation setup created in Java. The network is configured with a single Cloud layer on which the Cloud agent runs, eight Edge servers at the Edge layer, and numerous devices at the device layer. **Table 2** displays the configuration of the servers and devices. The tasks are generated randomly using a poisson distribution on devices with a capacity of 200 to 1000. Each device is divided into four categories: infotainment, computationally intensive, augmented reality, and health application. The distribution of tasks is based on the type of task and is random. While heavy computation gives more priority to CPU usage, delay, CPU count, and resource cost, health apps prioritize storage, storage cost, and delay. Priority in distribution is given to each type of device.

Table 2. Configuration of Edge Server

FACTORS	METRICS
Number of cores	16-20
Million Instructions per second	60k – 100k
Disk Size	10t-15tb
Virtual Machines	10-16
Operating System	Linux
Architecture	X86
Cost per BW	0.1-1.0
Cost per Sec	0.30-0.50
Cost per Mem	0.05-2.0
Cost per Storage	0.1-2.0

Prediction Results

Two measures, such as Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), are utilised for evaluating forecasting errors in order to determine the precision of the work. MAE is a measurement metric where the absolute error is derived using (3.13) and is the absolute value of the difference between the predicted value and the actual value.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} \left(y_i^p - y_i^a\right) \quad (4)$$

Percentage of MAE is used in determine the precision of proposed model. Lower the value of MAPE indicates the model is performing better in prediction. It is defined as in 5.

$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{y_i^p - y_i^a}{y_i^a}\right) * 100\% \quad (5)$$

In the above formula, predicted value is y_i^p, the actual value is y_i^a and N is the number of the predicted values in the dataset.

The B-LSTM is trained first with historical data from the device, and the prediction is tested for accuracy before being deployed for real-time predictions. The request made by the device and devices similar to their operation are taken as train data. **Fig 6** illustrates the accuracy of the training and tested data under 50 epochs. An accuracy rate of 90% is achieved at the minimum, and almost 96% is achieved at the maximum in the testing phase. This value is increased on further iterations with the real time prediction of task and load. Learning rate decide the gain ratio and accuracy of the model. A rate of 0 to 1 is taken for training the model. Here 0.0001 is taken as learning rate is taken as learning rate for B-LSTM, as the time taken to train the model is also considered while setting the parameters. **Fig 7** Shows the gain ratio comparison for different learning rates. As the number of iterations increases the gain ration also increases. On average gain ratio is achieved by 0.0001. Prediction based on this rate gives more accuracy in short time.

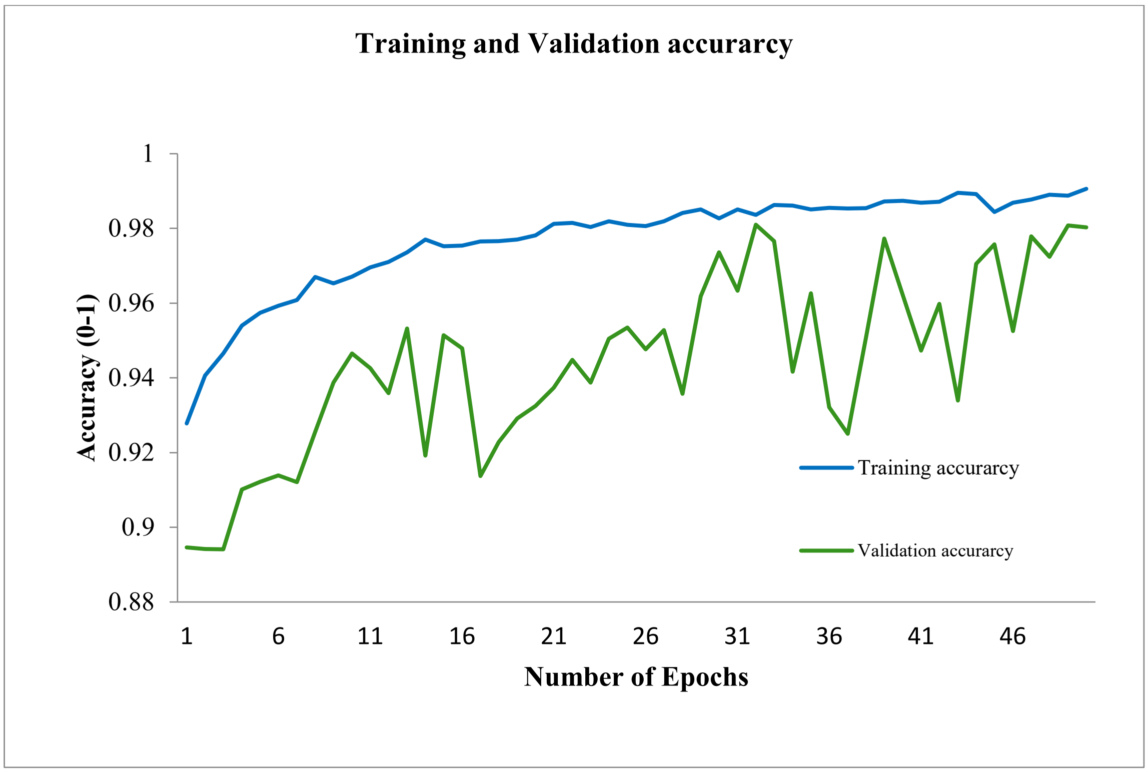

Fig 6. Training and Testing Accuracy**.**

Task Prediction Analysis

The tasks produced by terminal devices in an actual edge computing situation are closely tied to time and exhibit some degree of consistency and regularity. The task at t + 1 times is often predicted using a t times historical window. In the experiments, the history window is set to 50 and various optimization target thresholds. The B-LSTM prediction model can more accurately and thoroughly investigate the changing pattern of the historical data volume when the threshold value is set to minimal. However, this will result in a bigger prediction overhead and lengthen the B-LSTM model's training period.

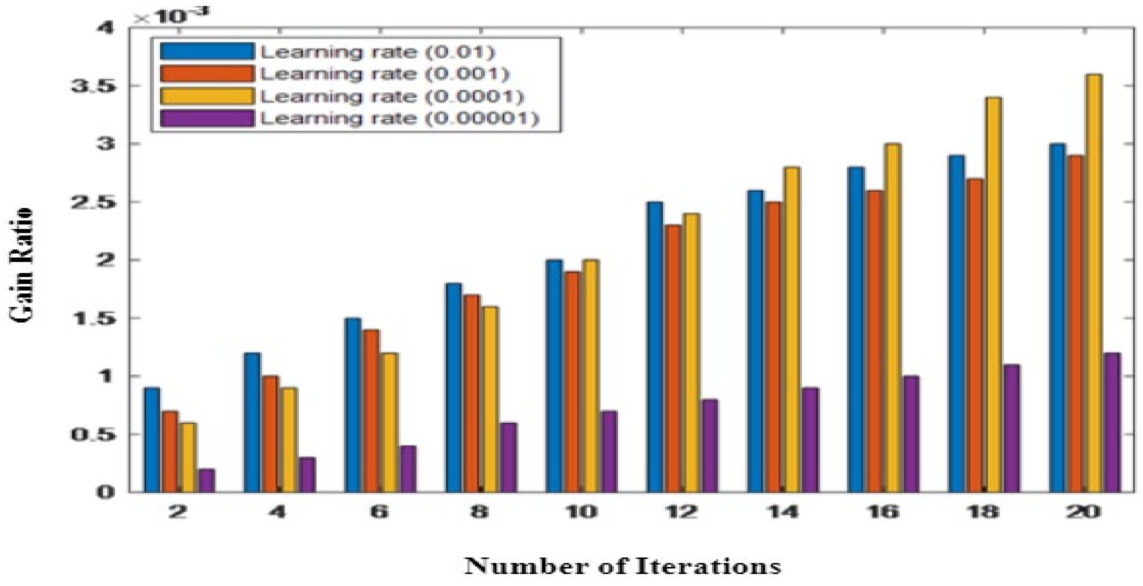

Fig 7. Analysis of Gain Ratios on Various Learning Rate and Iterations.

Existing works on LSTM, PROPHET, and ARIMA are taken for comparison on predicting the task creation in devices in order to assess the precision of our suggested model. Due to changes in the task type and nature of operation, the prediction was originally less accurate and varied greatly. Later on, the prediction started to escalate and eventually reached 90% in almost all models. On subsequent iterations, the results were essentially identical to the actual challenge, and our model consistently outperforms competing models in terms of accuracy, averaging 98% on average. When the devices abnormally generate a completely different request of service at certain periods, the model may suffer some losses in prediction. However, since proposed model resolves it using subsequent methods for uncertain task generation, it is negligible in a real-time situation. **Fig 8** compares the MAPE rate of the proposed model with those of other models. Since the early results are inaccurate, as previously indicated, the value is high and steadily decreases over time. When compared to other models, proposed model has high accuracy, as shown by the result, which continuously maintains a minimum MAPE of 2%. Offloading choices based on this forecast improve the device's QOE. By making this offloading decision based on our predictions, failure rate, delay, and processing time are reduced.

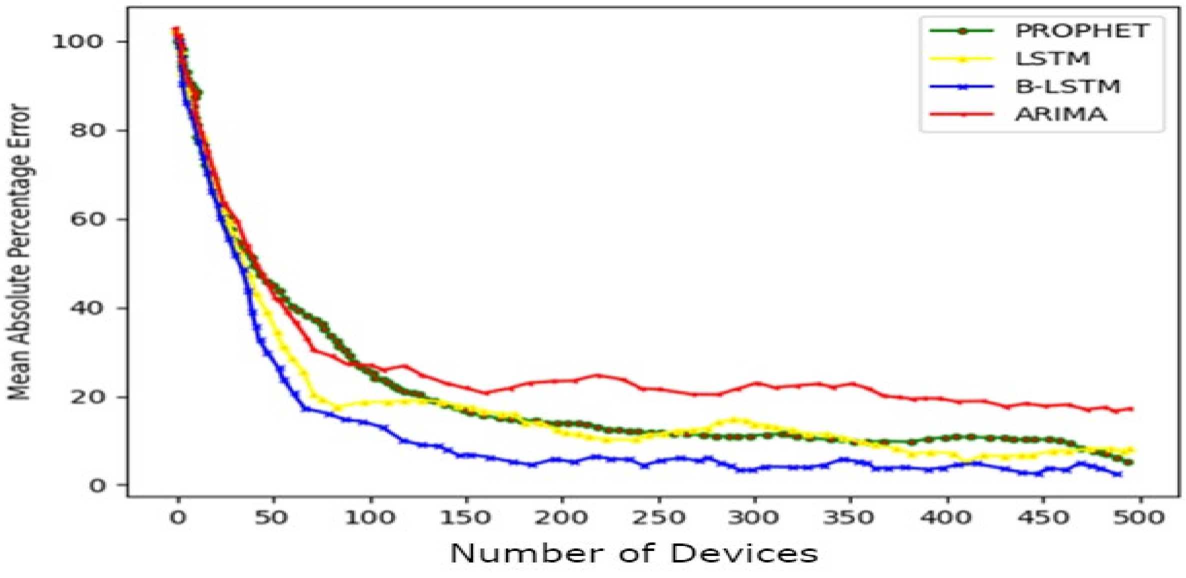

Fig 8. MAPE Comparison on Task Type Prediction Accuracy.

Performance Analysis

The proposed prediction model is assessed for its performance. The model is deployed with a multi-user and multiple edge servers in a distributed manner. Each device produces several tasks across various periods. Initial server loads are low, and each server has a unique configuration and set of resources. The devices' tasks are monitored and analyzed for prediction.

The task's QoE qualities are described by QOE_{att}, and devices have varying QoE attributes depending on the task request (x). The suggested model's objective is to determine the best offloading choice based on the qoe_{att} (x). The fundamentals of processing time, delay, failure rate, and device quality of experience are used to evaluate the model. Model is set with N = 20 mobile users, each of device performing multiple tasks. The device's local processing time is set at 3:75x10^{-7}s/bit and its power consumption at 3.55106 J/bit. Each task size will range from 10 to 35 MB. The bandwidth between a user and an edge server is set to 150 MB for both the uplink and the downlink, while actual bandwidth may vary depending on the condition of the network. An edge server'sCPU runs at a 9 108 cycle/s rate. The energy usage of the devices during transmission and reception is 1.60 106 J/bit. 100 episodes are taken to train the model.

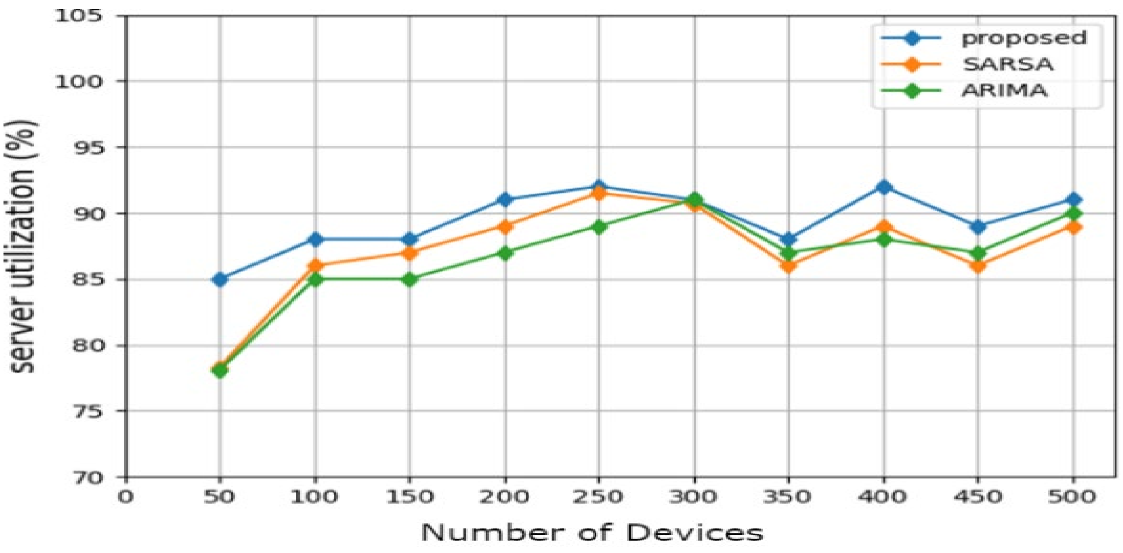

Fig 9. Comparison on Server Utilization of the Models**.**

Fig 9 shows the comparison of the percentage of server utilisation between different models. Server utilisation measures the amount of workload that is being distributed among the available servers over time. The balancing of load on the server is a crucial factor in making the offloading decision, as allocating tasks to an overloaded server will obviously increase the task queue. Unbalanced load allocation has downsides such as service failure and longer processing times. By anticipating the server's load condition and distributing resources based on its value, the load distribution is 96% balanced. Proposed model performed both this allocation and prediction while keeping an eye on dynamic network changes.

V. CONCLUSION

This research introduces a very effective way for transferring tasks to edge computing settings. It tackles the difficulties associated with traditional approaches by using sophisticated prediction models. The B-LSTM model is used to optimize task and server load prediction, leading to a substantial decrease in service delays and an enhancement in overall performance. The suggested Selective Objective Offloading Decision (SOOD) model, using the TOPSIS technique, exhibits exceptional performance by taking into account the distinct attributes of each device. The SOOD model surpasses previous models with a remarkable 98.4% enhancement in user Quality of Experience (QoE), offering a more customized and efficient offloading solution. Nevertheless, the erratic characteristics of some devices continue to provide a barrier, requiring more innovation in this field.

VI. FUTURE WORK

Prospective studies need to focus on improving the flexibility and resilience of offloading models in edge computing, particularly in managing unexpected workload patterns. By delving into advanced machine learning approaches like reinforcement learning or hybrid models that include various prediction algorithms, one might potentially find more robust answers. Moreover, including real-time feedback systems to constantly improve offloading choices according to changing environmental parameters would be advantageous. To enhance the applicability of the presented models in different real-world circumstances, it would be beneficial to broaden the research by include a wider variety of devices and distinct edge computing scenarios.

Data Availability
No data was used to support this study.

Conflicts of Interests
The author(s) declare(s) that they have no conflicts of interest.

Funding
No funding agency is associated with this research.

Competing Interests
There are no competing interests.

References

[1]. M. Eldred, C. Adams, and A. Good, "Trust Challenges in a High-Performance Cloud Computing Project," 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, vol. 53, pp. 1045–1050, Dec. 2014, doi: 10.1109/cloudcom.2014.21.
[2]. M. Armbrust et al., "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.
[3]. A. Jyoti, M. Shrimali, and R. Mishra, "Cloud Computing and Load Balancing in Cloud Computing -Survey," 2019 9th International Conference on Cloud Computing, Data Science &amp; Engineering (Confluence), pp. 51–55, Jan. 2019, doi: 10.1109/confluence.2019.8776948.
[4]. O. Ivanchenko, V. Kharchenko, B. Moroz, L. Kabak, and K. Smoktii, "Semi-Markov availability model considering deliberate malicious impacts on an Infrastructure-as-a-Service Cloud," 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), vol. 8, pp. 570–573, Feb. 2018, doi: 10.1109/tcset.2018.8336266.
[5]. F. Z. Benchara and M. Youssfi, "A new scalable distributed k-means algorithm based on Cloud micro-services for High-performance computing," Parallel Computing, vol. 101, p. 102736, Apr. 2021, doi: 10.1016/j.parco.2020.102736.
[6]. X. Pu, S. Jiang, and X. Zhang, "Cloud-Edge Collaborative Computation Offloading: A Deep Reinforcement Learning approach," 2022 International Conference on Networks, Communications and Information Technology (CNCIT), vol. 518, pp. 37–44, Jun. 2022, doi: 10.1109/cncit56797.2022.00015.
[7]. L. N. Mintarya, J. N. M. Halim, C. Angie, S. Achmad, and A. Kurniawan, "Machine learning approaches in stock market prediction: A systematic literature review," Procedia Computer Science, vol. 216, pp. 96–102, 2023, doi: 10.1016/j.procs.2022.12.115.
[8]. K. Kim, J. Lynskey, S. Kang, and C. S. Hong, "Prediction Based Sub-Task Offloading in Mobile Edge Computing," 2019 International Conference on Information Networking (ICOIN), Jan. 2019, doi: 10.1109/icoin.2019.8718183.
[9]. M. Zhao and K. Zhou, "Selective Offloading by Exploiting ARIMA-BP for Energy Optimization in Mobile Edge Computing Networks," Algorithms, vol. 12, no. 2, p. 48, Feb. 2019, doi: 10.3390/a12020048.
[10]. M. Goudarzi, Z. Movahedi, and M. Nazari, "Mobile Cloud Computing: A multisite computation offloading," 2016 8th International Symposium on Telecommunications (IST), vol. 48, pp. 660–665, Sep. 2016, doi: 10.1109/istel.2016.7881904.
[11]. M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioningand Scheduling Algorithm for Scientific Workflows on Clouds," IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 222–235, Apr. 2014, doi: 10.1109/tcc.2014.2314655.
[12]. L. N. T. Huynh, Q.-V. Pham, X.-Q. Pham, T. D. T. Nguyen, M. D. Hossain, and E.-N. Huh, "Efficient Computation Offloading in Multi-Tier Multi-Access Edge Computing Systems: A Particle Swarm Optimization Approach," Applied Sciences, vol. 10, no. 1, p. 203, Dec. 2019, doi: 10.3390/app10010203.
[13]. S. Dai, M. Liwang, Y. Liu, Z. Gao, L. Huang, and X. Du, "Hybrid Quantum-Behaved Particle Swarm Optimization for Mobile-Edge Computation Offloading in Internet of Things," Mobile Ad-hoc and Sensor Networks, pp. 350–364, 2018, doi: 10.1007/978-981-10-8890-2_26.
[14]. E. Darbanian, D. Rahbari, R. Ghanizadeh, and M. Nickray, "IMPROVING RESPONSE TIME OF TASK OFFLOADING BY RANDOM FOREST, EXTRA-TREES AND ADABOOST CLASSIFIERS IN MOBILE FOG COMPUTING," Jordanian Journal of Computers and Information Technology, no. 0, p. 1, 2020, doi: 10.5455/jjcit.71-1590557276.
[15]. J. Liu and Q. Zhang, "Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality," IEEE Access, vol. 7, pp. 11222–11236, 2019, doi: 10.1109/access.2019.2891113.
[16]. H. Qi, X. Mu, and Y. Shi, "A task unloading strategy of IoT devices using deep reinforcement learning based on mobile cloud computing environment," Wireless Networks, vol. 30, no. 5, pp. 3587–3597, Oct. 2020, doi: 10.1007/s11276-020-02471-4.
[17]. B. L. R, S. Murugan, and M. Balakrishnan, "Automatic Human Activity Detection Using Novel Deep Learning Architecture," EAI/Springer Innovations in Communication and Computing, pp. 441–453, 2024, doi: 10.1007/978-3-031-53972-5_23.
[18]. R. Nareshkumar, K. Agalya, A. Arunpandiyan, M. Vijayalakshmi, V. Ranjani, and A. Ramya, "An Effective Deep Learning based Recommender System with user and item embedding," 2023 International Conference on Artificial Intelligence and Knowledge Discovery in Concurrent Engineering (ICECONF), pp. 1–7, Jan. 2023, doi: 10.1109/iceconf57129.2023.10083578.