Journal of Information Systems and Telecommunication
Vol.13, No.3, July-September 2025, 243-255
http://jist.acecr.org
ISSN 2322-1437 / EISSN:2345-2773

Enhancing Computational Offloading for Sustainable Smart Cities: A Deep Belief Network Approach

Kaebeh Yaeghoobi[1*], Mahsa Bakhshandeh N.[2]

[1].Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran Iran
[2].Faculty of Engineering, Ale-Taha Institute of Higher Education, Tehran, Iran

Received: 12 Nov 2024/ Revised: 11 Sep 2025/ Accepted: 13 Oct 2025

Abstract

The use of mobile devices with limited processing power has surged in recent years, alongside the expansion of cloud and fog computing across various sectors. These devices can handle small to medium computing tasks, but they fall short when it comes to large-scale processes, making computational offloading a crucial solution. Cloud computing and fog computing provide an effective platform for offloading tasks from mobile devices. However, critical real-time applications necessitate a near-edge approach to managing the computational load. Significant challenges exist in optimizing response times for effective offloading in cloud computing. This research introduces a framework for predicting response times using Deep Belief Network (DBN) learning to enhance offloading performance. Implementing a DBN aims to minimize response times and resource consumption, thereby improving the overall efficiency of offloading processes. The framework is designed to predict response times accurately, ensuring timely completion of tasks and efficient use of resources. Simulation results using multiple models show that the use of DBN significantly reduces processing, response, and offloading times compared to other algorithms. Consequently, the DBN algorithm proves to be more efficient in predicting response times and enhancing offloading performance. By leveraging the capabilities of DBN, this framework provides a promising solution for optimizing computational offloading in cloud computing environments. This enhances the performance of mobile devices and ensures the reliability and efficiency of real-time applications, direct the way for more advanced and responsive computing technologies.

Keywords: Computational Offloading; Cloud Computing; Deep Belief Network; Response Time; Resource Management; Sustainable Smart Cities; Real-time Management.

1- Introduction

The proliferation of mobile devices has substantially increased computing demands, introducing new challenges in communication networks and resource provisioning. Due to their limited resources, mobile devices struggle with large-scale image processing and real-time conversion services [1]. Cloud computing technology helps mitigate these limitations; however, it is not applicable for real-time applications considering latency issues. Consequently, offloading computational tasks to independent platforms becomes a practical solution. For instance, the mobile cloud can provide maximum advantage for mobile video gaming and streaming [2].

Nevertheless, mobile cloud computing encounters challenges such as limited network bandwidth and offloading latency. Transmitting data from mobile devices to distant clouds consumes significant bandwidth, leading to traffic congestion and increased latency. Latency-sensitive applications require offloading to nearby locations, such as the nearest edge or mobile fog, to address these issues [3].

Cisco Systems introduced fog computing as an extension of cloud computing, bringing its capabilities to the network's edge. This extension benefits IoT services by supporting latency-intolerant mobile services. Numerous studies have focused on standardizing the computational offloading process at the edge or mobile fog, particularly in selecting mobile application units. Challenges related to offloading at

✉ **Kaebeh Yaeghoobi**

yaeghoobi@kntu.ac.ir

the mobile edge or fog include mobility, heterogeneity, and geographic distribution of devices.

As the digital world expands and network technologies evolve, complex services are emerging [4]. The generation of online applications featuring computing, communication, and intelligent capabilities continues to grow. Despite the growing power of current devices, they still struggle with tasks required for smart healthcare, augmented reality, intelligent car communication, and many smart city services. These applications often require another individual to execute tasks as a representative of the user's device, a technique known as process offloading [5]. Task disburdening is especially advantageous for Internet of Things and cloud computing requisition, facilitating interactions between edge devices or fog nodes and sensors and IoT nodes. Load shedding can be established on computational requirements, load balancing, energy management, and latency management [6].

In a data-rich world, mobile devices with limited resources can handle small-to-medium computations but struggle with high-level computations. Processing offloading is an effective solution to overcome this limitation. Recently, cloud computing has been recognized as a suitable platform for offloading tasks from mobile devices. However, the distance of cloud data centers from mobile devices increases network latency and affects the performance of real-time IoT applications.

For essential real-time applications, employing a near-edge network approach for computing offload is vital. Additionally, the primary controls for distributed mobile devices are heterogeneous in the offloading process of mobile computing. To overwhelm these contests, a deep learning-based response time prediction framework has been implemented to optimize offloading decisions near fog/edge or cloud nodes.

The objectives of this research are:

- Enhance Offloading Performance: Develop a deep learning-based framework to improve computational offloading efficiency.
- Minimize Prediction Error: Achieve the lowest discrepancy between actual and predicted response times using deep learning techniques.
- Boost Prediction Accuracy: Enhance the accuracy of response time predictions with the proposed deep learning method.

The paper is structured as follows: Section 2 covers related concepts and foundational research. Section 3 outlines the technical methodology, including the proposed method and framework. Section 4 analyses the proposed framework, presents results, and evaluates their theoretical implications. The final section discusses the results' implications and concludes with future trends and perspectives.

2- Background

This section explores concepts and metrics used in computational offloading, IoT middleware technologies, technologies that enhance fog computing tasks, and offloading methods in fog and cloud computing. The interplay between cloud, fog, and mobile computing models, concerning large computing resources, is analyzed. The literature review also covers computing resource allocation methods and achievements in cloud computing offloading.

Cloud computing resources are managed using virtualization technology. For example,[7] explains optimal virtual machine placement, examining distribution methods in cloud data centers. Most resource allocation mechanisms are designed for green computing. The DPRA allocation mechanism, discussed in [8], considers energy consumption of virtual and physical machines and data center air conditioning. A comparison of three schemes with DPRA shows energy savings, PM shutdowns, and reduced VM migrations.

In [9], a multi-objective optimization algorithm balances availability, costs, and performance for running big data applications in the cloud, outperforming conventional methods by reducing costs and achieving higher performance. However, the study focuses on big data applications.

In critical real-time applications, for example, patient control systems and intelligent transportation, mobile cloud computing offloads large tasks while maintaining quality standards [10]. A mobility-aware resource allocation architecture, Mobihat, provides efficient scheduling but does not study the impact of mobility on delay and response times for real-time mobile services.

Offloading mobile edge computing with multiple users, based on TDMA and OFDMA, is introduced in [11]. The TDMA-based method reduces mobile energy consumption, while the OFDMA hybrid model transforms into TDMA, defining a discharge priority function for optimal resource allocation.

The optimal computational offloading framework for DNNs is presented in [12], considering mobile batteries and cloud resources. This method evaluates energy consumption and execution time.

In [13], battery life of nearby mobile devices is used to select discharge positions. A non-interactive game model, maximizing player payoffs, reduces response times. The Nash equilibrium is obtained through the game model and indirect induction method, evaluated for response time, end-user benefit, and memory usage. Yang et al. [14] address high implementation delays among mobile devices and fog nodes using queuing theory. Data rate and power consumption are selected as decision parameters, formulating a multi-objective optimization problem to decrease transmission energy consumption, power, and

cost, determining the probability of discharge for all mobile devices.

A survey on stochastic-based offloading methods in different computing environments, including mobile cloud, edge, and fog computing, is proposed in [15]. The classification is divided into Markov chain, Markov process, and hidden Markov models, discussing open issues and future challenges.

In [16], a multi-objective optimization model addresses time and energy consumption of mobile users and edge server resource utilization. An edge-cloud joint offloading method, based on the evolved Strength Pareto algorithm, is effective and efficient for scenarios with multiple mobile users and heterogeneous edge servers.

An offloading architecture, combining intelligent computing with AI, is presented in [17]. Considering mobile task data size and edge node performance, a load shedding and task transfer algorithm optimize edge computing offloading. Experiments show reduced task delay by increasing data and subtask execution.

Du et al. [18] address offloading in a cloud-cloud environment, supporting a heterogeneous model to consider task communication cost asymmetry. They prove the NP-hard nature of the problem and design an efficient algorithm for an optimal solution, evaluated through a PageRank-based program in a controlled cloud edge setting.

An adaptive wireless resource allocation strategy for computational offloading, under a three-layer edge cloud framework, is studied in [19]. Modeling the offloading process at the minimum block level of allocable wireless resources adapts to vehicular scenarios and evolves in the 5G network. The proposed value density function measures cost-effectiveness and energy saving. Numerical results show the designed algorithm achieves significant running time and energy savings, with superior performance compared to benchmark solutions.

An autonomous computational offloading framework is presented in [20] for time-consuming programs, addressing control model challenges for managing computing load. Various simulations, including deep neural networks and hidden Markov models, are performed. Results show the hybrid model fits the problem with near-optimal accuracy for discharge decisions, delay, and energy consumption predictions. MAPE is used for discharge, collection, and processing for decision making. The proposed method outperforms local computing and offloading in latency, energy consumption, network utilization, and execution cost.

In [21], minimizing average task execution time in edge systems, considering job request heterogeneity, application data pre-storage, and base station cooperation, is addressed. A mixed integer nonlinear programming (MINLP) problem is formulated and addressed using decomposition theory. The GenCOSCO algorithm improves service quality and computational complexity. For fixed service cache configurations, the FixSC algorithm derives evacuation strategies, with simulations showing significant task execution time reductions.

Peng et al. [22] propose three multi-objective evolutionary algorithms to tackle the computing offloading challenges in IoT for edge and cloud networks. They developed a constrained multi-objective load calculation model that accounts for time and energy consumption in mobile environments. Drawing inspiration from the push and pull search (PPS) framework, they introduced three algorithms (PPS-NSGA-II, PPS-SPEA2, and PPS-SPEA2-SDE) that integrate population-based search with flexible constraint control. These algorithms were tested using multi-task, multi-user scenarios across various IoT devices. The results demonstrated their effectiveness and superiority.

Other research presents a user-centered joint optimization offloading scheme designed to minimize the weighted costs of time delay and energy consumption. The mixed-integer nonlinear programming problem is addressed using a particle swarm optimization algorithm that incorporates 0-1 and weight improvement techniques. Simulation results indicate higher performance in delay, energy consumption, and cost [23].

In [24], a computation offloading scheme via mobile vehicles in a cloud-IoT network is proposed. Sensing devices generate tasks and transmit them to vehicles, which then decide whether to compute the tasks locally, on a MEC server, or at a cloud hub. The offloading decision is based on a utility function that considers energy consumption and transmission delay, using a learning-based approach. Experimental results show that this solution maximizes rewards and reduces delay.

Based on the research discussed, various techniques can be adopted for cloud computing offloading, depending on priorities. This research proposes using a response time prediction model based on deep learning to determine the optimal offloading position. The impact on delay and energy efficiency will be evaluated to improve offloading performance by minimizing the error between actual and predicted response times.

3- Methodology

A mobile fog node expands the capabilities of fog and mobile cloud computing models by offering a localized system to minimize potential delays and execution times while maintaining continuous and direct communication in conjunction with the cloud data center. The proposed model, depicted in Figure 1, encompasses three offloading positions: the cloud data center, adjacent mobile station, and mobile fog. This setup is supported by the LTE hierarchical architecture and the Wi-Fi intra-network reference model, situating the mobile fog at the network's edge. Access points and access point controllers operate as mobile fog nodes.

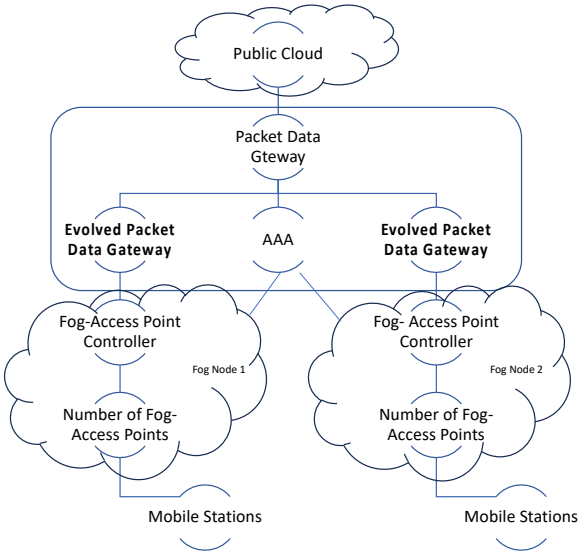

Fig. 1 Mobile Fog System Model for Computational Offloading - Verification and confirmation of Mobile Stations is Achieved by 3GPP AAA via Extensible Authentication Protocol-Authentication and Key Agreement(EAP-AKA) over Internet Key Exchange version 2 (IKEv2)

Within this architecture, the mobile edge/fog is represented by the fog-1 node, the mobile fog by the fog-2 node, and the public cloud serves as the third offloading position, referred to as the cloud node. Communication within the fog is enabled by the Evolved Packet Core, which provides the Evolved Packet Data Gateway.

Access points not only facilitate communication between mobile stations but also offer cloud services such as, Network as a Service (NaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). IEEE Ethernet interfaces connect access points to access point controllers, while IEEE 802.11 WLAN interfaces link mobile stations to access points. The access point controller manages block code migration, overseeing memory, processing, I/O, and networking capabilities to sustain mobile cloud services. Hence, the access point controller similarly serves as a fog network controller. In Figure 1, fog-enabled access points are labeled as "fog-access points," and access point controllers are designated as "fog-access point controllers." Mobile station authentication is conducted by the 3GPP AAA via EAP-AKA over IKEv2, with the verification and validation vector derive through the shared home server unit in the LTE network. The data network gateway, which handles access to user equipment or mobile stations and virtual machines (VMs), has evolved into a packet data gateway. The top module, the public cloud, functions as a traditional delivery network, providing pervasive and scalable services accessible via the web using both mobile and static devices.

3-1- Unloading Node Process

This section details the offloading process based on the previously described model, with a focus on the fog/mobile edge. In critical real-time applications, nodes such as public cloud and mobile fog and mobile edge are physically dispersed to deliver services to mobile cloudlets, which are resource-limited mobile stations. Due to the dynamic nature of these applications, request times are unknown and random, with variable response times, making it challenging to identify the optimal offloading node.

To tackle this issue, a deep learning-based approach is recommended. This approach learns from the request history and response times of nodes to predict future response times. The node with the lowest predicted response time is then selected for offloading. The relationship between the computing requirements of cloud or fog nodes and the response time of virtual machines is complex.

Predicting workload data patterns is challenging due to their non-consecutive nature. Therefore, aggregated workload data characteristics of VMs are used instead of single VM data for prediction purposes. A deep learning model can better determine workload data dispersions based on inherent data characteristics, outperforming simpler models. This preference is due to the deep model's ability to learn complex relationships between workload data features. Although structurally similar to a Multi-Layer Perceptron (MLP), a Deep Belief Network (DBN) has a diverse training method, allowing it to address gradient fading effectively.

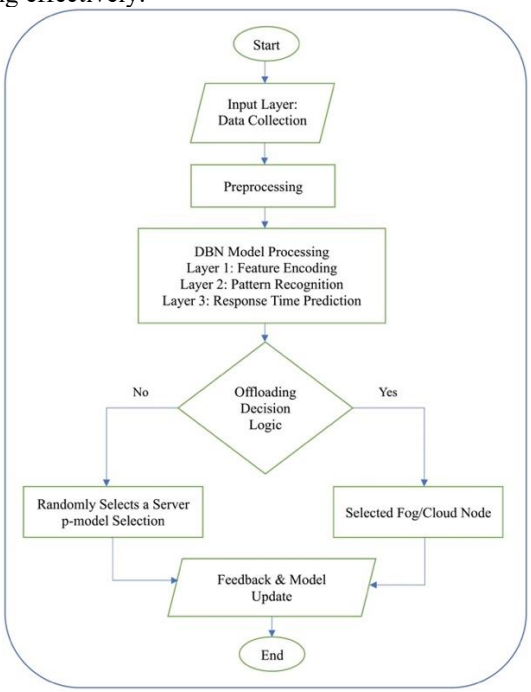

Fig. 2 Flowchart of the DBN-based offloading decision process, integrating predictive modelling, fallback selection via p-model, and feedback-driven model updates for sustainable smart city applications.

Figure 2 illustrates the complete workflow of the proposed DBN-based computational offloading system for smart city environments. The process begins with data collection from mobile devices and virtual machines, including historical request patterns and aggregated workload characteristics. After preprocessing and feature extraction, the data be used for DBN step, which performs multi-layer encoding and pattern recognition to predict future response times of candidate nodes. Based on these predictions, the system attempts to select the node with the lowest latency for offloading. If due to unpredictable workload patterns or insufficient confidence no suitable node is identified, the system activates a fallback mechanism using the p-model, which randomly selects a server based on predefined probability. The final stage involves task execution and feedback logging, which continuously refines the DBN model for future decisions.

3-2- Deep Belief Network (DBN)

A Restricted Boltzmann Machine (RBM) can extract features and recreate data entry, in spite of that, it struggles with gradient blurring. To address this, multiple RBMs can be combined with a classifier to form a Deep Belief Network (DBN). This method, known as greedy layer-by-layer unsupervised pretraining, involves training the DBN two layers at a time, treating each pair of layers as an RBM. In this architecture, the hidden layer of one RBM acts as the input layer for the subsequent RBM. The training process starts with the initial RBM, whose outputs are fed into the next RBM, and this sequence continues until the output layer is reached. Through this process, the DBN identifies inherent data patterns, functioning as an advanced multi-layer feature extractor. A unique aspect of this network is its ability to learn the complete structure of the input at each layer, similar to a camera gradually focusing an image.

Finally, labels are applied to the resulting patterns in the DBN. The DBN is subsequently fine-tuned through supervised learning using a small set of labeled samples, with minor changes to weights and biases leading to a marginal increase in accuracy.

The proposed approach includes a deep belief network with one-layer neural network. This method employs an unsupervised approach to extract more robust and helpful features from VM workload data. By increasing the hidden layers in the DBN, the error gradient is significantly amplified before being minimized. Training is conducted using an unsupervised greedy layer-wise method. To further optimize, the DBN's top layer utilizes a standard sigmoid regression. Future request predictions are generated by analyzing response times in terms of bandwidth *(B)*, memory *(M)*, and processing capability *(P)*.

As presented in Figure 3, inputs to the DBN model include the bandwidth, memory and processing capability of entire requests, along with the recent workload of all VMs. These data cover actual response times discovered over various time spans. For each node, the trained DBN models predict response times, with input values normalized between 0 and 1. The core layer's units equal the sum of the VMs in the cloud and the time slots.

$$Number\ of\ Units = VM \times TI \quad (1)$$

Where:

VM represents the number of virtual machines.

TI represents the number of time intervals.

This simple yet effective formula helps determine the total number of units required based on the given parameters.

Alternatively, a supervised approach with a precisely configured logistic regression layer can be employed to label the data and predict the workload of a VM.

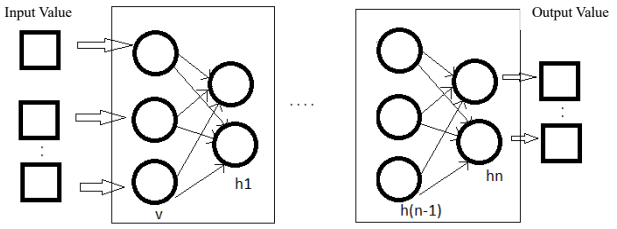

Fig. 3 Stacks before RBM Training

Initially, the standard binary RBM is modified to a Gaussian-Bernoulli RBM. The visible unit biases in the RBM energy function are adjusted to include quadratic bias terms [3]. An example of a load shedding decision session is shown in Table 1. The Energy function and Conditional Probability Distribution are conveyed in following way:

$$\mathrm{E}(\mathrm{x},\mathrm{h}|\theta) = \sum_{i=1}^{X} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j=}^{H} b_j h_j - \sum_{i=1}^{X}\sum_{j=1}^{H} \frac{v_i}{\sigma_i} h_j w_{ij} \quad (2)$$

$$P(h_i|x;\theta) = \delta\left(\sum_{i=1}^{X} w_{ij}x_i + b_j\right) \quad (3)$$

$$P(x_i|x;\theta) = N(\sigma_i \sum_{i=1}^{X} w_{ij}x_i + a_j, \sigma_i^2) \quad (4)$$

Table 1: Description of symbols

Symbol	***Description***
μ	mean
σ^2	variance
σ	standard deviation
P	probability
E	expectancy
X	observable variables
H	common hidden space of variables
W	linear mapping coefficient
B	bias

In this context, the Gaussian distribution's probability distribution function is represented by $N(\mu,\sigma^2)$, where μ is the mean, and σ^2 is the variance vector. Hinton's training method outlines the prediction process as follows:

Unsupervised Training: The RBN visible and hidden layer are trained. The RBM input comprises a request section and a response time dataset. θ is the only non-continuous parameter in the RBM.

Layer Inheritance: Each visible layer in RBM inherits and utilizes the extracted features of the preceding RBM as its input. This process is repeated for subsequent RBMs, with the parameter θ retained for the next and initial RBM.
Input to Logistic Regression: The regression layer is trained using labelled data in a supervised manner; and input of that is the output of the final RBM.
Supervised Training: The θ parameters are trained and adjusted using the backpropagation (BP) algorithm.
The deep belief network-based response time prediction method leverages edge/cloud computing to accurately determine whether to offload computations to a neighbouring node, an edge/fog node, or a cloud node. To handle the unpredictability of resource availability in edge/fog and cloud nodes, the proposed offloading procedure leverages the technique of RBM learning.
To begin the substantial data volumes and the demand for real-time applications, particularly in the e-health sector, a near-edge network approach for offloading computations is recommended. This strategy addresses the primary controls for distributed mobile devices, easing the offloading process in mobile and heterogeneous computing environments. A deep learning-based response time prediction framework has been developed to enhance computational offloading performance, determining the optimal offloading target, whether it's a nearby fog/edge node, an adjacent fog/edge node, or a cloud node. Additionally, the Restricted Boltzmann Machine (RBM) learning technique is utilized to handle the variability of resource availability.
In this study, the DBN model was trained using aggregated workload data collected from simulated virtual machines operating under diverse conditions. The training process involved unsupervised pre-training of Restricted Boltzmann Machines (RBMs) followed by supervised fine-tuning using labeled response time data. Training was conducted on a standard CPU-based computing environment, which, was sufficient for the scale and complexity of the dataset used. The total training time varied depending on the configuration, typically ranging from 30 minutes to 2 hours. Once trained, the model was deployed for inference on edge servers, where its lightweight architecture enabled real-time prediction without significant computational overhead. This setup demonstrates that even without specialized hardware, the DBN-based offloading strategy remains practical and effective for mobile and fog-based environments.

4- Result and Analysis

This section examines the performance of the proposed models. The simulation results integrate real mobility tracking, server datasets, and model implementation on actual machines. Subsequent sections will explore the performance benefits of DBN-based models using three probability distributions (uniform, normal, and exponential) to achieve accurate results.

4-1- Data Collection

To simulate mobile node movements, a dataset of vehicle movements in Rome was utilized, as referenced in [25]. This dataset comprises coordinates of 320 taxis collected over 30 days, including their coordinates, date, time, and GPS location. Mobility tracking treats any movement as a point in time to check server or dump time, rather than studying user mobility. Each movement is modeled as an interaction with a mobile edge computing server. Processing times are obtained from real servers (CPU usage), involving around 150 data servers (over 1 billion rows). With e very movement, a server is selected from the dataset, its utilization is checked, and an unloading decision is made based on the model's recommendation.
The evaluation spans more than five days (5000 rows of movements). An evacuation decision is made every minute, resulting in over 1000 evacuation decisions, ensuring the proposed models' behavior is observed over an extended period. The DBN-based response time prediction method leverages edge/cloud computing to determine whether to offload computations to a neighboring node, an edge/fog node, or a cloud node.
Given the challenges posed by large data volumes and real-time applications, particularly in the e-health sector, a near-edge network approach was recommended for offloading computations. The proposed RBM learning technique addresses the randomness of resource availability.
Figure 4 distribution of server usage probabilities across all servers in the dataset. The data generally follows a normal distribution, illustrating typical CPU utilization patterns observed during simulation.

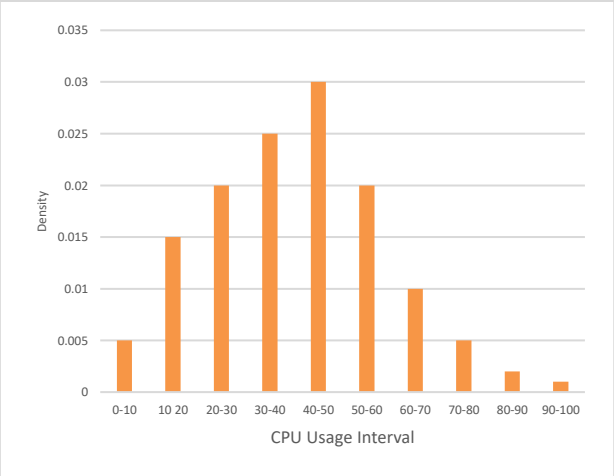

Fig. 4. CPU usage distribution of servers (CPU unit is percentage and Density is J)

Table 2 sample load shedding decision session, showing CPU consumption values for selected mobile edge computing servers at specific geographic positions and time intervals.

Table 2: Dataset Sample Used in the Experiment. (ID: xx6, Motion Time Interval: 10 Seconds)

Position	*Machine*	*CPU Consumption*
X=41.8911, Y=12.49073	M_xx39	51
X=41.89905, Y=12.4899	M_xx36	47
X=41.8994, Y=12.48940	M_xx41	20
X=41.8994, Y=12.489401	M_xx41	37

4-2- Evaluation

This section focuses on simulating and evaluating the proposed evacuation rules across various variables. The primary aim is to observe the models' behavior under different conditions, allowing generalization to parameters such as quality of service and response time concerning computational load.

MATLAB software is chosen for the simulation, which can perform process-based discrete event simulation. The "Advance Mode" is selected for the probability distribution of the random variable *X*, including time (processing). In the simulation, a resource actually is a mobile edge computing server *k* that is modelled and can advertises its processing time *Xk*. A process is a mobile node that modelled to traverses the mobile edge computing servers and checks latency of each server based on the processing time. Initially, we consider *n=5,* means having five mobile edge computing servers. The processing time *X* follows a normal distribution (50 ms to 10 ms), a uniform distribution in the interval [0-1], and a binominal distribution of 50 J/mol. MATLAB has generated incidental variables following the determined apportionment.

At every initiation, a node begins polling the mobile edge computing servers consecutively, starts with server one. At this step, the proposed approaches are utilized to choose a mobile edge computing server. The important parameters in processing time are waiting time, delay and total delay. Additionally, based on the program types, the range of processing time differs from 100 milliseconds to 800 seconds, and in intervals of 10 milliseconds to 30 milliseconds. Therefore, various ranges for parameter *X* can be considered derived from the proposed models, which producing similar outcomes as observed in the experiment dataset. Table 3 shows the values and range of parameters in the simulation test.

The main approach used in the simulation involves comparing values obtained from other studies, random values, the nearest server (immediate loading), and a method from the same family of algorithms proposed in this work. This evaluation is limited to comparisons between different models, including the random and probabilistic model *(p)*. These approaches are compared to the superior option, where the server or time with the minimum value is chosen.

Table 3: Simulation Parameters Values for all Methods

Parameters	*Value / Range of Values*
X	N(10, 50) & U(0, 1)
No. of mobile nodes	1000
N	{3, 5, 10}
P for p-model	0.8
R	{0, 0.25, 0.5, 1}
θ	{30, 40, 50, 60} {0.3, 0.4, 0.5, 0.6} {20, 30, 40, 50, 60}
C	{1, 2, 3, 4, 5, 20, 30} {0.1, 0.2, 0.3, 0.4} {1, 10, 15, 20, 30, 40}

The reasons for adopting this approach are as follows: Primarily, this research emphasizes data decision-making and task offloading. Additionally, deep learning algorithms inherently differ from traditional algorithms, especially when the decision maker lacks complete information. Thus, the approach to optimality is the main analysis for evaluating these algorithms. Optimization is suitable when all server information is available to the decision maker, facilitating the mobile node in determining the ideal offloading location. Ultimately, these algorithms are implemented in sequence, complicating direct comparisons with other algorithms.

In this setting, in the absence of offloading rules, the mobile node will likely choose the first available mobile edge computing server. For edge computing load, such an offloading method is optimal for task offloading. So, the p-model method is utilized as a fallback technique. In the p-model, each server is assigned a loading probability, set to *p=0.8*. During each user move, each server has a probability *p=0.8* of being selected to load the job. In this experiment, increasing *p* intensively the probability of selecting the first server for loading. Consequently, the p-model replicates the scenario where the mobile node chooses the nearest servers that is closest edge servers due to the higher probability *p=0.8*.

When evaluating the actual dataset, if a server is preferred (server is chosen for loading) the process stops; if no server is preferred, the last server is chosen. A server is randomly preferred for each user to offload the work in the random selection model.

The results of all models are compared with values obtained from the proposed model, where the server with the shortest processing time is chosen for each unloading session. Models that are closer to the optimal value demonstrate superior performance in offloading decisions. The optimal

model is achieved by choosing the server with the shortest processing time for each load sequence.

4-3- Results

The simulation results evaluate the performance of the proposed DBN-based offloading model across multiple dimensions, including execution time, server usage, energy efficiency, and successful offloads. The evaluation spans three distinct probability distributions for the processing time variable X: normal, uniform, and exponential. Each distribution reflects different real-world workload scenarios in mobile edge computing environments.

Across all simulations, the DBN-based model consistently demonstrates superior performance compared to benchmark algorithms such as Delay Tolerant Offloading (DTO), Best Choice Problem (BCP), Cost-based Optimal Task (COT), Quality-Aware Odds, Random selection, and the p-model. The proposed method achieves lower average execution times, reduced CPU usage, and higher rates of successful offloads under varying resource constraints.

Figures 5 through 13 present comparative results for each distribution scenario. These include average processing times, server utilization, and the number of effective offloads under different CPU thresholds. The DBN model shows strong alignment with the optimal model, particularly in scenarios where resource availability is dynamic and unpredictable. This confirms the model's ability to make accurate offloading decisions and maintain system efficiency under diverse conditions.

Performance Analysis with Normal Distribution

As illustrated in Figure 5, when the processing time X follows a normal distribution, the proposed DBN-based algorithm achieves the shortest execution time among all evaluated methods. The average execution time for computational discharge is approximately 40 milliseconds, outperforming DTO, BCP, COT, and the p-model algorithms.

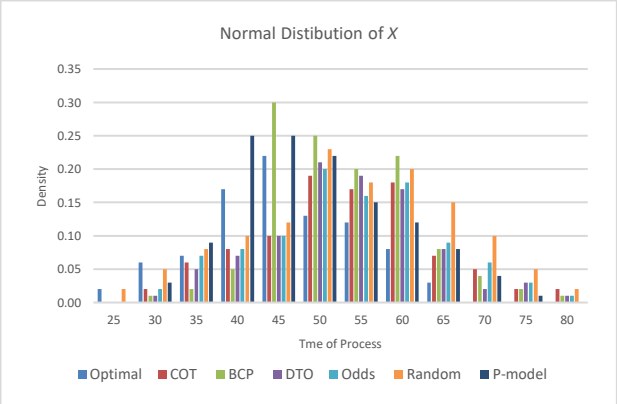

Fig. 5 Simulation Results for All Models in Case of *X* Normal Distribution.

The figure also reveals a significant overlap between the DBN model and the optimal model, indicating that the DBN's predictions closely approximate ideal offloading decisions. In contrast, models such as the p-model and random selection exhibit higher variance and longer processing times. The BCP model achieves a processing time of 46 milliseconds, which is lower than the p-model and random approaches but still less efficient than the DBN. These results validate the effectiveness of the DBN-based offloading strategy in minimizing latency and optimizing resource allocation in mobile edge computing. The model's ability to learn from historical workload patterns and predict response times contributes to its superior performance across varying conditions.

The results in Figure 6 reveal that the variation between the optimal model and DBN model is significantly smaller than the variation detected with other models. Notably, for models other than the DBN, the optimal threshold for each experiment k is generally close to the average processing time of 50 milliseconds. For example, in the DTO model and COT model, the thresholds generated for n=5 are {40, 42, 43, 46, 50}, all near the average processing time.

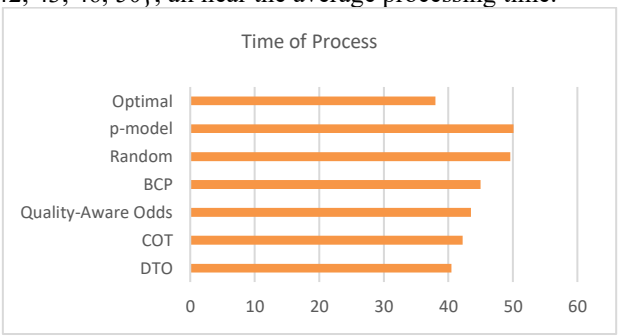

Fig. 6 Average Processing Time for Different Models with *X* Normal Distribution.

Using these optimal thresholds as a reference, the initial threshold value for the Odds method is set to 50, with performance evaluated for various values. The results, indicate the effective performance of the Odds method. This performance can be credited to the high likelihood of choosing a server with a processing time under 50 milliseconds. Thus, by setting a threshold value close to the average processing time, a shorter processing time is achieved for unloading the computational load.

Furthermore, the results demonstrate better performance for the BCP method compared to the p-models and Random method. The BCP evacuation policy is more likely to achieve the shortest processing time, leading to a lower average processing time than other models. This increased likelihood results in a lower expected processing time compared to the random and p models

Significantly, while the probability of selecting the best server is assumed to be similar in the BCP and Odds models, the defined threshold in the Odds model enhances performance by ensuring quality-aware decisions when

examining mobile edge computing servers. The main conclusion from these results is that the proposed method, referred to as the optimal model, achieves a shorter processing time than other methods, thereby reducing response time and improving the performance of computational offloading in cloud computing.

Performance Analysis with Uniform Distribution

In the initial results, the random variable X followed a normal distribution. To achieve more accurate findings, we conducted an additional simulation with X uniformly distributed within the interval *[0-1] (Figure 7)*. This range represents server usage, such as CPU utilization, where a value of 0.5 indicates 50% CPU usage. We applied similar steps to all models, as in previous experiments.

In the DTO model, the delay coefficient initially began at $r=0$, with results for other r values presented subsequently. For the cost-based optimal task model, an ideal threshold was identified for each cost value in the second set. Specifically, for c = 0.2, evaluations determined the optimal threshold to be 0.3. The cost interpretation is similar to the normal distribution scenario: a higher cost (smaller threshold V) signifies a greater need for shorter processing times.

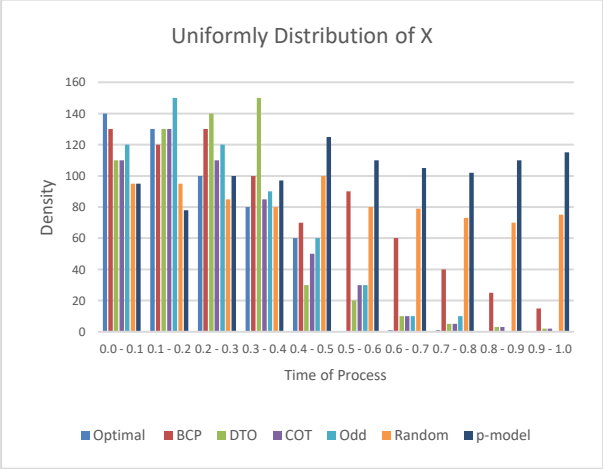

Fig. 7 Simulation Results for All Models in Case of X Uniformly Distribution.

In the quality-aware Odds model, the threshold was set to 0.5, yielding a 42% probability of selecting a server with $X=0.5$. Though the BCP model shares this probability, setting the threshold notably improved the Odds model's performance. Figures 7 and 8 show that model performance aligns closely with results from the normal distribution scenario. DTO and COT models remain top performers, with deep belief network-based models coming closer to optimality compared to random and p models.

As illustrated in Figure 8, the average execution time for various algorithms, including the proposed method based on the deep belief network, has been evaluated. The results demonstrate that the proposed method achieves a shorter execution time compared to other methods, indicating a more efficient response to computational offloading in mobile edge computing.

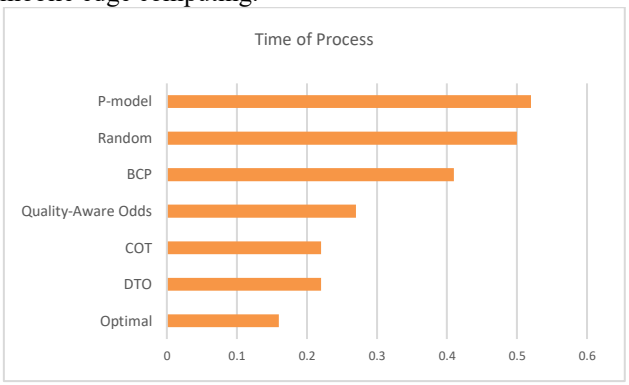

Fig. 8 Average Processing Time for Different Models with Uniform X Distribution.

Performance Analysis with Exponential Distribution

Figure 8 demonstrates that the proposed algorithm achieves an execution time of approximately 0.15 milliseconds, which is shorter compared to other methods. On the other hand, the p-model algorithm exhibits the longest execution time due to the consideration of a threshold value for selecting servers. These results suggest that the deep belief network (DBN) method provides superior response times for computational offloading in mobile edge computing, attributed to its layered approach.

Besides normal and uniform distributions, this experiment also included an exponential distribution with a mean of 50. The same procedural steps were followed as in the previous distributions. Initially, the delay coefficient in the DTO method was set to $r=0$, with results for other r values subsequently presented. The results under these conditions are shown in Figures 9.

In the Cost-based Optimal Task model, the figures depict the optimal threshold values V corresponding to each cost value. For this simulation, the cost was initially set to 20, with the optimal threshold determined to be 45.81, resulting in the lowest simulated expectation of X among other values. Performance across various cost values is also demonstrated. The cost interpretation aligns with scenarios where X follows normal and uniform distributions: a higher cost (smaller threshold V) indicates an increased demand for shorter processing times.

In the quality-aware Odds method, the threshold was set to 50, resulting in a 44% probability of selecting a server with X=50. The results in Figures 9 and 10 indicate that the proposed model's performance is consistent with the results obtained when X follows normal and uniform distributions. The DBN-based method consistently outperforms other algorithms, demonstrating the best performance and closest

proximity to optimality compared to the random and p-models.

Figure 9 demonstrates that the proposed method with exponential distribution achieves a lower execution time compared to other methods. This distribution effectively guides server selection for mobile edge calculations, showing that the deep belief network-based method provides a faster response for computational offloading in mobile edge computing than other algorithms.

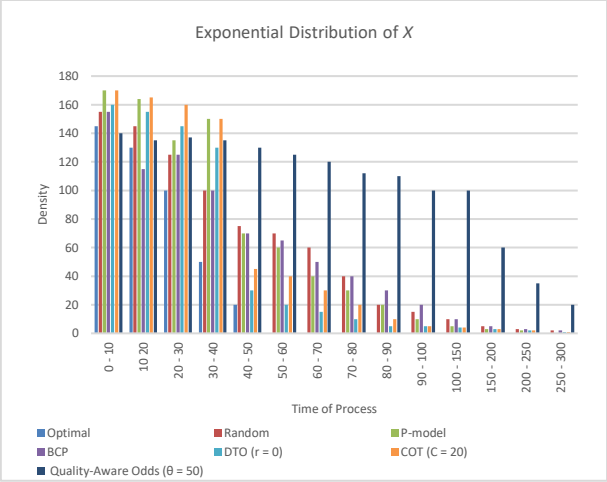

Fig. 9 Simulation Results for All Models in Case of *X* Exponential Distribution.

Figure 10 illustrates the average response time for different methods with exponential distribution. The proposed method has a significantly lower response time, approximately 10 milliseconds, compared to other algorithms. This demonstrates that the proposed method surpasses other approaches in reducing response time for computational offloading in mobile edge computing.

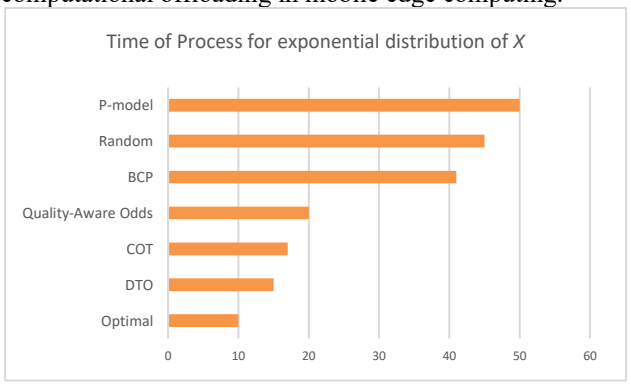

Fig. 10 Average Processing Time for Different Models With X Exponential Distribution.

Server Usage and Energy Efficiency

Figure 11 illustrates the average server usage recommended by each model. The DTO and COT models show results closest to the proposed method, with DTO performing better than the others by an absolute difference of 23 units compared to the proposed method. The findings indicate that the proposed method has a lower average server consumption than the other methods, meaning it consumes less energy for mobile edge calculations.

Additionally, the proposed method, based on the deep belief network, demonstrates a shorter average unloading time compared to other algorithms. Consequently, this suggests that the response time for computational offloading in mobile edge computing is more efficient with the proposed method than with others.

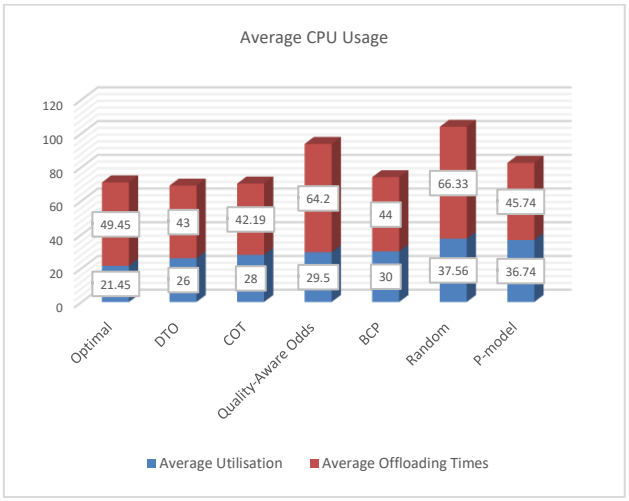

Fig. 11 Average CPU Usage and Average Computational Drain Time by each Model

Server Consumption and Successful Offloads

Figure 12 illustrates the average server consumption for the proposed method compared to other solutions. The proposed deep belief network method demonstrates a lower average server consumption, indicating that it not only reduces the response time for computational offloading but also optimizes server usage. This results in lower overall server consumption compared to other algorithms.

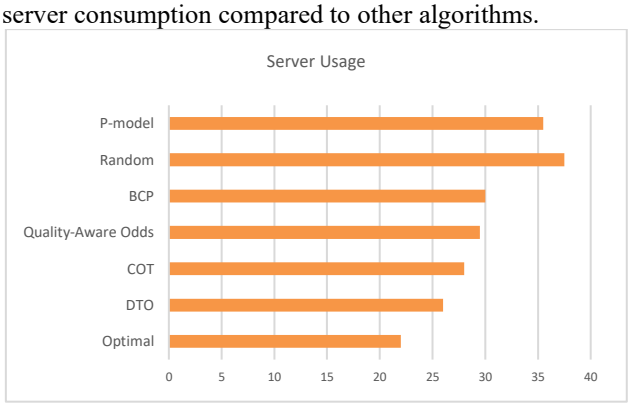

Fig. 12 Average Usage of Servers for Different Algorithms.

Result presented the average server consumption for the proposed method compared to other solutions. The proposed deep belief network method demonstrates lower average server consumption, indicating that it not only

reduces response time for computational offloading but also optimizes server usage, resulting in lower overall server consumption compared to other algorithms.

Beyond average server utilization, we compare performance based on the number of effective offloads for each model. An effective offload refers to unloading decisions that meet specific requirements set by each model. To assess this, we assume three different mobile edge computing programs (x, y, and z) each with distinct needs. For example:

- Program x requires less than 10% CPU utilization.
- Program y requires less than 20% CPU utilization.
- Program z requires a server with less than 30% CPU utilization.

If an offload occurs for a server with usage less than 10%, it is considered a successful offload for program x.

Figure 13 illustrates the effective offloads for various resource demands across entire methods. The proposed deep belief network-based method achieves the highest number of successful offloads in these three cases, with values of 102, 463, and 1887 successful offloads, respectively.

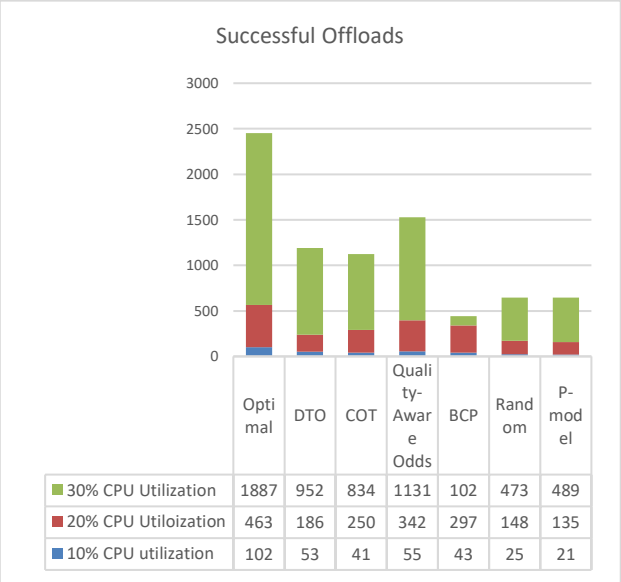

Fig. 13 Number of Effective Discharges for each Model Based on Various Threshold Values.

4-4- Discussion

The simulation results for various methods indicate that the presented models generally exhibit a time complexity of *O(n)* at worst, both in terms of time and space. If each model's condition is met on server number n, the mobile node will visit server n. For the DTO, COT, and Quality-aware Odds models, a pre-observation step involves generating thresholds. This step is presumed to be executed a single time by the service provider, external to the mobile node, although it can be implemented within the mobile node if necessary. For example, computing the threshold at the mobile node in the Odds and DTO methods requires *O(n)* time complexity. The COT method requires more time to calculate the threshold, depending on the likelihood distribution. Merely a sole operation is essential for a (uniform) distribution, while a normal distribution requires integration estimation with a time complexity no greater than $O(n^2)$.

Regarding space complexity, the BCP model does not require additional space for data storage, resulting in a space complexity of *O(n)*. This also applies to other models, provided the training step is performed outside the mobile node. If the training step is conducted locally at the mobile node, only the probability distribution parameters need to be stored. For a uniformly distributed *X,* the maximum and minimum values are stored, while for exponentially distributed *X*, the μ mean and σ^2 standard deviation are required. Previous results showed that the time complexity of the proposed method based on a deep belief network (DBN) is *O(1)*, the lowest complexity for predicting time and improving computational offloading performance in mobile edge computing.

Analyzing the execution time and server consumption across different algorithms reveals that the proposed method is more efficient in performing the computational offloading process. The results indicate that the proposed model is completely independent and lightweight for implementation in the mobile node, outperforming other compared solutions. The DBN-based method requires less processing time for computational offloading and task execution, with lower CPU consumption than other solutions. This makes it suitable for managing computational offloading of resources, compressing, or delaying limited tasks.

A practical scenario that highlights the effectiveness of the proposed DBN-based offloading mechanism involves a mobile user engaged in augmented reality (AR) navigation within a smart city. AR applications are latency-sensitive and require rapid processing of environmental data, user location, and graphical overlays. In such a context, the DBN model predicts the response times of available fog and cloud nodes based on historical workload patterns and real-time system conditions. By selecting the node with the lowest predicted latency, the system ensures that AR content is rendered and delivered with minimal delay, thereby preserving user experience and application responsiveness. In cases where no optimal node is identified, the fallback mechanism ensures continuity by probabilistically selecting a viable server. This dynamic and adaptive offloading strategy demonstrates the model's potential to support real-time, resource-intensive mobile applications in complex urban environments.

5- Conclusion

The principal aim of this research is to enhance computational offloading performance in mobile edge computing. To achieve this, we have employed a computational analysis method based on the deep belief network (DBN), incorporating various deep learning features to improve the evacuation process. By adding specific steps to the computational evacuation process, we aim to reduce server consumption, increase process speed, and decrease response time to computational requests.

In this study, the deep belief network algorithm has been utilized to further optimize computational offloading, making it suitable for various cloud computing applications, including mobile edge computing. The proposed algorithm focuses on reducing execution time for requests and increasing the number of successful offloads within the mobile edge computing system. By combining different distribution functions and the core features of the DBN algorithm, our method seeks to enhance efficiency and the volume of computational offloading.

Our approach to computational offloading on the server side is designed to provide a solution with low response time, ultimately reducing time complexity and energy consumption. It is crucial to employ the appropriate method to perform this process efficiently. Incorrect algorithms for computational offloading in cloud computing can lead to increased energy consumption and decreased successful offloads. Timely offloading reduces server-side energy consumption and increases efficiency, highlighting the importance of an accurate response time prediction solution to improve computational offloading performance in mobile edge computing.

A detailed examination of our results indicates that the proposed algorithm effectively improves computational offloading in mobile edge computing. This algorithm requires less time to execute offloading processes and respond to requests from mobile nodes. The number of requests handled by the servers does not increase response time, thereby reducing the duration of computational offloading. Compared to Delay Tolerant Offloading (DTO), Best Choice Problem (BCP), Cost-based Optimal Task (COT), and p-model algorithms, our method demonstrates shorter average processing times for computational offloading and request responses, achieving optimal results for the evaluated dataset. The proposed method outperforms other methods in terms of time complexity, energy consumption, processing time, CPU usage, average offload time, and the number of successful offloads.

While the proposed algorithm sometimes exhibits longer processing times for specific requests, overall performance in processing time, resource utilization, average server usage, successful offloads, and computational offload time is superior in improving computational offloading in mobile edge computing. By balancing accuracy and speed, our method effectively reduces response time and increases the number of successful offloads.

Future research should evaluate the proposed method across various cloud computing systems, applications, and datasets to fully explore its efficiency and applicability. Additionally, further studies can investigate other neural network algorithms, such as long short-term memory and convolutional neural networks, to enhance offloading performance in mobile edge computing. Meta-heuristic algorithms may also be considered to address the NP-hard nature of computational offloading problems, aiming to reduce complexity and increase successful offloads. Finally, developing solutions that require minimal processing and computing resources, while considering available resource consumption, will lead to more efficient computational offloading and increased successful offloads.

References

[1] A. Mahdavi and A. Ghaffari, “Embedding Virtual Machines in Cloud Computing based on Big Bang–Big Crunch Algorithm,” *Journal of Information Systems & Telecommunication (JIST)*, p. 305, 2019.

[2] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, Jan. 2013, doi: 10.1016/J.FUTURE.2012.05.023.

[3] Md. G. R. Alam, M. M. Hassan, Md. Z. Uddin, A. S. Almogren, and G. Fortino, “Autonomic computation offloading in mobile edge for IoT applications,” *Future Gener. Comput. Syst.*, vol. 90, pp. 149–157, 2019, [Online]. Available: https://api.semanticscholar.org/CorpusID:52899499

[4] P. Boopathy *et al.*, “Deep learning for intelligent demand response and smart grids: A comprehensive survey,” *Comput Sci Rev*, vol. 51, p. 100617, Feb. 2024, doi: 10.1016/J.COSREV.2024.100617.

[5] I. Abdullaev, N. Prodanova, K. A. Bhaskar, E. L. Lydia, S. Kadry, and J. Kim, “Task Offloading and Resource Allocation in IoT Based Mobile Edge Computing Using Deep Learning,” *Computers, Materials and Continua*, vol. 76, no. 2, pp. 1463–1477, Aug. 2023, doi: 10.32604/CMC.2023.038417.

[6] H. Naseri, S. Azizi, and A. Abdollahpouri, “BSFS: A Bidirectional Search Algorithm for Flow Scheduling in Cloud Data Centers,” *Journal of Information Systems and Telecommunication (JIST)*, vol. 3, no. 27, p. 175, 2020.

[7] D. Seddiki, F. J. Maldonado Carrascosa, S. García Galán, M. Valverde Ibáñez, T. Marciniak, and N. Ruiz Reyes, “Enhanced virtual machine migration for energy sustainability optimization in cloud computing through knowledge acquisition,” *Computers and Electrical Engineering*, vol. 119, p. 109506, Oct. 2024, doi: 10.1016/J.COMPELECENG.2024.109506.

[8] L.-D. Chou, H.-F. Chen, F.-H. Tseng, H.-C. Chao, and Y.-J. Chang, “DPRA: Dynamic Power-Saving Resource Allocation for Cloud Data Center Using Particle Swarm Optimization,” *IEEE Syst J*, vol. 12, no. 2, pp. 1554–1565, 2018, doi: 10.1109/JSYST.2016.2596299.

[9] H. Wang, S. Cao, H. Li, L. Yan, Z. Guo, and Y. Gao, "Multi-objective joint optimization of task offloading based on MADRL in internet of things assisted by satellite networks," *Computer Networks*, vol. 254, p. 110801, Dec. 2024, doi: 10.1016/J.COMNET.2024.110801.

[10] T. Tsokov and H. Kostadinov, "Dynamic network-aware container allocation in Cloud/Fog computing with mobile nodes," *Internet of Things*, vol. 26, p. 101211, Jul. 2024, doi: 10.1016/J.IOT.2024.101211.

[11] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading," *IEEE Trans Wirel Commun*, vol. 16, no. 3, pp. 1397–1411, 2017, doi: 10.1109/TWC.2016.2633522.

[12] Q. Wang, S. Guo, J. Liu, and Y. Yang, "Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing," *Sustainable Computing: Informatics and Systems*, vol. 21, pp. 154–164, Mar. 2019, doi: 10.1016/J.SUSCOM.2019.01.007.

[13] S. C. Ghoshal *et al.*, "VESBELT: An energy-efficient and low-latency aware task offloading in Maritime Internet-of-Things networks using ensemble neural networks," *Future Generation Computer Systems*, vol. 161, pp. 572–585, Dec. 2024, doi: 10.1016/J.FUTURE.2024.07.034.

[14] S. Yang, D. Kwon, H. Yi, Y. Cho, Y. Kwon, and Y. Paek, "Techniques to Minimize State Transfer Costs for Dynamic Execution Offloading in Mobile Cloud Computing," *IEEE Trans Mob Comput*, vol. 13, no. 11, pp. 2648–2660, 2014, doi: 10.1109/TMC.2014.2307293.

[15] X. Xu, Q. Huang, X. Yin, M. Abbasi, M. R. Khosravi, and L. Qi, "Intelligent Offloading for Collaborative Smart City Services in Edge Computing," *IEEE Internet Things J*, vol. 7, no. 9, pp. 7919–7927, Sep. 2020, doi: 10.1109/JIOT.2020.3000871.

[16] T. Tang, C. Li, and F. Liu, "Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning," *Comput Commun*, vol. 209, pp. 78–90, Sep. 2023, doi: 10.1016/J.COMCOM.2023.06.021.

[17] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, and M. S. Hossain, "Intelligent task prediction and computation offloading based on mobile-edge cloud computing," *Future Generation Computer Systems*, vol. 102, pp. 925–931, Jan. 2020, doi: 10.1016/J.FUTURE.2019.09.035.

[18] M. Du, Y. Wang, K. Ye, and C. Xu, "Algorithmics of Cost-Driven Computation Offloading in the Edge-Cloud Environment," *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1519–1532, 2020, doi: 10.1109/TC.2020.2976996.

[19] L. Tan, Z. Kuang, J. Gao, and L. Zhao, "Energy-Efficient Collaborative Multi-Access Edge Computing via Deep Reinforcement Learning," *IEEE Trans Industr Inform*, vol. 19, no. 6, pp. 7689–7699, Jun. 2023, doi: 10.1109/TII.2022.3213603.

[20] A. Shakarami, A. Shahidinejad, and M. Ghobaei-Arani, "An autonomous computation offloading strategy in Mobile Edge Computing: A deep learning-based hybrid approach," *Journal of Network and Computer Applications*, vol. 178, p. 102974, Mar. 2021, doi: 10.1016/J.JNCA.2021.102974.

[21] S. Zhong, S. Guo, H. Yu, and Q. Wang, "Cooperative service caching and computation offloading in multi-access edge computing," *Computer Networks*, vol. 189, p. 107916, Apr. 2021, doi: 10.1016/J.COMNET.2021.107916.

[22] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained Multiobjective Optimization for IoT-Enabled Computation Offloading in Collaborative Edge and Cloud Computing," *IEEE Internet Things J*, vol. 8, no. 17, pp. 13723–13736, 2021, doi: 10.1109/JIOT.2021.3067732.

[23] Z. N. Samani and M. R. Khayyambashi, "Reliable resource allocation and fault tolerance in mobile cloud computing," *Journal of Information Systems and Telecommunication (JIST)*, vol. 7, no. 2, pp. 96–109, 2019.

[24] J. Long, Y. Luo, X. Zhu, E. Luo, and M. Huang, "Computation offloading through mobile vehicles in IoT-edge-cloud network," *EURASIP J Wirel Commun Netw*, vol. 2020, no. 1, p. 244, 2020, doi: 10.1186/s13638-020-01848-5.

[25] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," 2014.